A Thesis

entitled

Inverse Modeling: Theory and Engineering Examples

by

Rahul Rama Swamy Yarlagadda

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the
Master of Science Degree in Mechanical Engineering

Dr. Efstratios Nikolaidis, Committee Chair

Dr. Vijay Devabhaktuni, Committee Co-Chair

Dr. Matthew Franchetti, Committee Member

Dr. Mehdi Pourazady, Committee Member

Dr. Patricia R. Komuniecki, Dean
College of Graduate Studies

The University of Toledo
December 2015

Copyright 2015, Rahul Rama Swamy Yarlagadda

This document is copyrighted material. Under copyright law, no parts of this document may be reproduced without the expressed permission of the author.

An Abstract of
Inverse Modeling: Theory and Engineering Examples

by

Rahul Rama Swamy Yarlagadda

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the
Master of Science Degree in Mechanical Engineering

The University of Toledo
December 2015

Over the last two decades inverse problems have become increasingly popular due to their widespread applications. This popularity continuously demands designers to find alternative methods, to solve the inverse problems, which are efficient and accurate. Using effective techniques that are both highly accurate and of low computational cost is of highest priority.

This thesis presents a method for solving inverse problems through Artificial Neural Network (ANN) theory. This thesis also presents a method to apply Grey Wolf optimizer (GWO) algorithm to solve inverse problems. GWO is a recent optimization method demonstrating great results. Both of the methods are then compared to traditional methods such as Particle Swarm Optimization (PSO) and Markov Chain Monte Carlo (MCMC).

Four classical engineering design problems are used to compare the four methods' performance. The results from the engineering design problems show that the GWO outperforms other methods in terms of efficiency and accuracy. The error is comparable among the proposed ANN method and PSO method, while the latter has better computational efficiency.

To my mother (Padma) and my loving sister (Yoshitha) for their unconditional love,
support and encouragement.

Acknowledgments

I am deeply indebted to my advisors Dr. Efstratios Nikolaidis and Dr. Vijay Devabhaktuni whose motivation and support helped me in the completion of my master's degree study and research. I would also like to thank Dr. Matthew Franchetti and Dr. Mehdi Pourazady for being on my thesis committee and for their valuable feedback.

The financial support from the MIME department and the civil department in the form of assistantship is gratefully acknowledged. I would like to thank Dr. Cyndee Gruden for her opportunity as research assistant and encouragement.

I would like to thank my dearest friend Ruthwik Reddy Junuthula for his support to complete my research. I also thank my friend Anthony Dietrich for his support at all times.

I am also very thankful to my friends and colleagues, Neel Kamal, Sai, Jabbir, Abhishek, Colin, Hem, Yogesh, Jahnavi, Divya, Praveen, and Miracle, at the University of Toledo for their love and friendship.

Above all, I would like to thank my mother and my sister who with their love, and encouragement have made all this possible.

Contents

Abstract	iii
Acknowledgments	v
Contents	vi
List of Tables	ix
List of Figures	x
List of Abbreviations	xii
List of Symbols	xiii
1 Introduction	1
1.1 Definition of an Inverse Problem	1
1.1.1 Model Space and Data Space	1
1.2 Significance of Inverse Problems	4
1.3 Outline of Thesis	4
2 Literature Survey	5
2.1 Inverse problems: Background	5
2.2 Significance of Multimodal Solutions	8
2.3 Review of Monte Carlo methods	8
2.4 Review of Particle Swarm Optimization	9

2.5	Review of Artificial Neural Networks	10
2.6	Review of Grey Wolf Optimizer	12
3	Methods of Approach	13
3.1	Markov Chain Monte Carlo	13
3.2	Particle Swarm Optimization	16
3.3	Grey Wolf Optimizer	20
3.4	Artificial Neural Networks	25
4	Examples	29
4.1	Dynamic Vibration Absorber	29
4.2	Tension Compression Spring	32
4.3	Pressure Vessel Design (PVD)	35
4.4	Ten Bar Truss Problem	38
5	Conclusions and Future Work	42
	References	43
A	MATLAB code for MCMC	52
A.1	Dynamic Vibration Absorber	52
A.2	Tension Compression Spring	54
A.3	Pressure Vessel Design	56
A.4	Ten Bar Truss Problem	58
B	MATLAB code for PSO	62
B.1	Dynamic Vibration Absorber	62
B.2	Tension Compression Spring	64
B.3	Pressure Vessel Design	65
B.4	Ten Bar Truss Problem	67

C	MATLAB code for GWO	69
C.1	Dynamic Vibration Absorber	69
C.2	Tension Compression Spring	72
C.3	Pressure Vessel Design	76
C.4	Ten Bar Truss Problem	79
D	MATLAB code for ANN	83
D.1	Dynamic Vibration Absorber	83
D.2	Tension Compression Spring	89
D.3	Pressure Vessel Design	94
D.4	Ten Bar Truss Problem	99

List of Tables

4.1	Comparison of performance of the techniques for DVA	32
4.2	Comparison of performance of the techniques for Tension/Compression Spring	35
4.3	Comparison of performance of the techniques for Pressure Vessel design .	38
4.4	Input summary	39
4.5	Comparison of performance of the techniques for the Ten-Bar Truss problem	41

List of Figures

1-1	Tension/Compression Spring	2
1-2	Conventional division of a model into a forward and an inverse model . .	3
2-1	Number of publications on Inverse Problems over the time	6
2-2	Positron emission tomography- CT Scanner	6
2-3	Inverse Problem of ECG Imaging	7
2-4	Number of publications on Markov Chain Monte Carlo over the time . .	9
2-5	Number of publications on Particle Swarm Optimization over the time .	10
2-6	Number of publications on Artificial Neural Networks over the time . . .	11
3-1	MCMC technique: Random Walk	14
3-2	Flowchart for the MCMC technique	15
3-3	A bird flock	16
3-4	A murmuration of starlings	17
3-5	Flowchart for the PSO technique	19
3-6	Grey wolves hunting behaviour	20
3-7	Hierarchy of grey wolves	21
3-8	Flowchart for the GWO technique	24
3-9	A typical neural network with multiple inputs and outputs	25
3-10	Flowchart for the ANN technique	28
4-1	Cylindrical DVA attached to helicopter blades	30
4-2	Tuned damper system	31

4-3	Tension/Compression Spring	33
4-4	Suspension spring in an automobile	34
4-5	Pressure Vessel	36
4-6	Schematic of a Pressure Vessel	37
4-7	Arawaka truss bridge in Japan	39
4-8	Ten-Bar Truss with loads applied	40

List of Abbreviations

ANN	Artificial Neural Network
CG	Conjugate Gradient
DVA	Dynamic Vibration Absorber
GWO	Grey Wolf Optimizer
IP	Inverse Problem
MCMC	Markov Chain Monte Carlo
PDE	Partial Differential Equation
PSO	Particle Swarm Optimization
PVD	Pressure Vessel Design
SD	Steepest Descent
TBT	Ten Bar Truss

List of Symbols

α		alpha wolves
β		beta wolves
δ		delta wolves
ϵ		strain tensor
ω		omega wolves
σ		stress tensor

Chapter 1

Introduction

1.1 Definition of an Inverse Problem

1.1.1 Model Space and Data Space

Let M be a physical system under study. For instance, M may be a galaxy to an astrophysicist, earth to a geophysicist, or a dynamic system such as a car body. The scientific method for the study of a physical system consists of three steps [1].

1. Describe the physical system: find the least set of model parameters whose values completely define the system.
2. Develop a forward model: find the physical laws that enable us to predict the system response to a given input. The response is quantified by a set of observable parameters, such as the vibratory amplitude of the front door of the car body.
3. Inverse modeling: from the desired response to infer the values of the model parameters.

The first two steps are mainly inductive, but the last step is deductive. Therefore, it is challenging to make explicit the rules of thinking that we follow in the first two

steps. On the other hand, one can explain the rules in the third step. For example, one can estimate the model parameters by minimizing the sum of the square errors of the model predictions.

Model Space: An Example

The choice of the model parameters to be used to describe a system is generally not unique.

A tension/compression spring S (shown in Fig. 1-1) is analyzed in a laboratory. To describe the spring's elastic properties, it is possible to use the tensor C of elastic stiffnesses relating stress, σ , to strain, ε , at each point (x) of the spring:

$$\sigma(x) = C(x) \cdot \varepsilon(x) \quad (1.1)$$

Alternatively, it is possible to use the tensor $S(x)$ of elastic compliances relating strain to stress,

$$\varepsilon(x) = S(x) \cdot \sigma(x), \quad (1.2)$$

where the tensor S is the inverse of C . The use of stiffnesses or of compliances is completely equivalent, and there is no 'natural' choice.

Figure 1-1: Tension/Compression Spring [2]

Data Space

To obtain information on model parameters, we have to perform a physical experiment and measure some observable parameters. For example, consider a nuclear physicist interested in the structure of an atomic particle. The observations may consist of a set of measurements of the flux of particles diffused at different angles for a given incident particle flux. For a geophysicist interested in understanding the earth's deep structure, observations may consist of a recorded set of seismograms at the ground surface. We can thus arrive at the abstract idea of data space, which consists of all conceivable instrumental responses.

“Physical theories allow us to make predictions: given a complete description of a physical system, we can predict the outcome of some measurements. This problem of predicting the result of measurements is called the forward problem” [1]. The inverse problem consists of using desired response to infer the values of the parameters that characterize the system. Figure 1-2 shows a conventional division of a model into a forward and inverse model.

Figure 1-2: Conventional division of a model into a forward and an inverse model [3]

1.2 Significance of Inverse Problems

Many important real-world problems are Inverse Problems (IPs). These include medical imaging, non-destructive testing, oil and gas exploration, land-mine detection and process control. For instance, in the exploration for oil and gas, we need to measure the structure of the earth's interior from the observations made at the surface. Usually, an explosion is created and the subsequent shock-waves coupled with their reflections are used to construct a model of the earth's structure.

At their simplest level, IPs are concerned with finding data about the interior of a body from data which is available on its surface. The mathematical equipment needed to solve IPs is largely founded on mathematical analysis and uses tools from functional analysis, function theory, spectral theory, the theory of PDEs, and integral equations [4, 5, 6, 7]. Moreover, an analyst needs tools of numerical analysis and scientific computing to solve many applied problems [8].

1.3 Outline of Thesis

The remaining thesis is structured as follows. Chapter 2 presents a detailed literature review of the inverse problems and traditional methods to solve them. These methods include numerical optimization. Chapter 3 presents an ANN approach, GWO application along with particle swarm optimization and Markov chain Monte Carlo methods to solve inverse problems. Chapter 4 discusses in detail and solves four examples of classical engineering problems using the four methods mentioned. Finally, Chapter 5 concludes the thesis suggests the future work.

Chapter 2

Literature Survey

2.1 Inverse problems: Background

In the last few decades, inverse problems have become increasingly popular and experienced explosive growth due to their widespread applications in acoustics, astrophysics, biomedical and civil engineering, geophysics, instrumentation, mechanical engineering, meteorology, oceanography, optics, radar, radio astronomy, and spectroscopy [9, 10, 11, 12, 13, 14]. The author searched for the number of published literature on Inverse Problems, Markov Chain Monte Carlo, Particle Swarm Optimization, Artificial Neural Networks, and Grey Wolf Optimizer using www.worldcat.org. Worldcat is the world's largest library catalog [15]. Figures 2-1 to 2-6 show the information from the website. The data on the Y-axes are the number of publications and on X-axes are the years. Figure 2-1 shows the increase in the number of publications on inverse problems with time. The rate increases significantly after 1985. Figure 2-3 shows the inverse modeling of an ECG imaging system. Also, positron emission tomography (PET) and single photon emission computed tomography (SPECT, shown in Fig. 2-2) are two recent imaging methods that require solving inverse problems [4, 16].

Figure 2-1: Number of publications on Inverse Problems over the time [15]

Figure 2-2: Positron Emission Tomography-Computer Tomography scanner [16]

Figure 2-3: Inverse Problem of ECG Imaging [17]

In the early 1920s, V. A. Ambartsumian first established the field of inverse problems in the astrophysics while he was trying to solve the Sturm-Liouville problem [18]. In the next few decades, the field has extended to many other prominent areas of science and technology. Albert Tarantola was the first to introduce the formulation of inverse problems in [19]. All the necessary constraints required for the formulation of an inverse problem are in his publication [20].

2.2 Significance of Multimodal Solutions

Many real-world engineering applications involve optimizing objective functions with multimodal behavior, i.e.; they have multiple global and local optima [21, 22]. Identifying these global and local optima will help the designer obtain multiple innovative solutions and understand the nature of the problem [23]. For example, in the design of a car, an ideal design may be the one that minimizes cost and fuel consumption while maximizing the overall performance. However, these goals are conflicting: one car design may have high performance at a high cost, while an alternative low-cost design would considerably increase the fuel consumption. Therefore, finding multiple solutions enables a designer to select the best solution.

2.3 Review of Monte Carlo methods

Keilis-Borok and Yanovskaya (1967) were the first to apply Monte Carlo simulation methods to the solution of inverse problems [24]. Since Monte Carlo methods require time-consuming computations, the introduction of computers catalyzed the interest in these methods [25]. Researchers from varying areas applied Monte Carlo methods at the end of the twentieth century. Geman and Geman [1984] advocated the use of simulated annealing to Bayesian image restoration [26]. However, they did not describe how to extend this idea to a general inverse problem.

Figure 2-4 shows the number of publications on Markov Chain Monte Carlo methods with time. Marroquin et al. (1987) implemented a similar approach to Geman and Geman's but, they used the Metropolis algorithm [27]. This algorithm enables one to sample from a probability distribution without having to derive a closed-form expression for the distribution. Recently, K. Mosegaard [1995], proposed the random sampling of solutions to inverse problems [28]. However, this method produces a single solution. As discussed earlier, real life inverse problems require multimodal

solutions, which is a challenge for the application of Monte Carlo methods to solve inverse problems.

Figure 2-4: Number of publications on Markov Chain Monte Carlo over the time [15]

2.4 Review of Particle Swarm Optimization

J. Kennedy et al. (1995) introduced the particle swarm optimization in 1995, which is inspired by social behavior of bird flocks [29]. It is a stochastic optimization method in that it uses random numbers similarly to the Genetic Algorithms [30]. It soon gained interest after its introduction, and it has been applied in many research areas. C. R. Mouser et al. (2004) were the first to apply particle swarm optimization method to solve a structural dynamics problem [31]. The authors argue that it is possible to apply evolutionary optimization methods such as genetic algorithm and particle swarm optimization to inverse problems. They showed that particle swarm optimization outperformed the genetic algorithm. Also, particle swarm optimization is much easier to configure than a genetic algorithm [32]. However, they did not

perform a multimodal optimization; instead they found a single solution to the structural problem. At the same time, S. L. Ho et al. (2004) proposed a particle swarm optimization method in the area of electromagnetics and mathematics [33]. While this method involves multiple objectives, it is unimodal optimization.

Figure 2-5: Number of publications on Particle Swarm Optimization over the time [15]

2.5 Review of Artificial Neural Networks

Due to generalization capability and faster learning ability of ANNs, they are being used in computer aided modeling applications [34]. The effectiveness of ANNs depends on the number of hidden neurons and training samples [35]. ANNs are being used for function approximation and data classification despite their poor extrapolation capability.

Recently, ANNs are being used to diagnose several cancers such as neuroblastoma, breast cancer, prostate cancer and skin cancer [36, 37, 38, 39]. ANNs are being applied

in various categories like data processing, robotics, and function approximation [40]. Due to their learning ability and data classification, ANNs are also used to identify handwritten zip codes by the U.S. Postal Service [41]. ANNs play a vital role in estimating the measurements of air quality and water quality using available data for years [42]. ANNs are known for their generalization capability. Unlike conventional methods, ANNs can model the data with less complexity in programming [43, 44].

Figure 2-6: Number of publications on Artificial Neural Networks over the time [15]

S.R.H. Hoole (1991) published the first paper on the solution of an inverse problem to the field of electromagnetics [45]. The author used a back-propagation algorithm to train the neural network and obtained promising results [46]. However, the author did not impose any constraints on the problem, which may have affected the accuracy of the solution.

2.6 Review of Grey Wolf Optimizer

Grey Wolf Optimizer is a recent optimization method developed by S. Mirjalili et al. (2014). It was inspired by the way that the Grey Wolves find, encircle, and hunt their prey. S. Mirjalili et al. (2014) argues that the algorithm yields competitive results when benchmarked with Particle Swarm Optimization, and Genetic algorithms [47]. However, there is no literature on the application of Grey Wolf Optimization to inverse problems.

Chapter 3

Methods of Approach

The four methods mentioned in the previous chapter are considered in this thesis to solve the inverse problems.

1. Markov Chain Monte Carlo (MCMC)
2. Particle Swarm Optimization (PSO)
3. Grey Wolf Optimizer (GWO)
4. Artificial Neural Networks (ANN)

The next subsections explain the formulation of the inverse problems and the methods for solving them.

3.1 Markov Chain Monte Carlo

MCMC is an integration method using Markov chains [48, 49, 50]. Figure 3-2 presents the steps of MCMC. The method draws a sample in each step from a proposed distribution and then evaluates the fitness function of the system of interest [51]. A fitness function may be the vibratory amplitude or the mass of a structure. In each iteration of MCMC, a candidate design $\tilde{X}$ is chosen by sampling from a proposal

distribution, and is accepted with a probability proportional to the fitness function of the system [48, 51]. For the next iteration, $\tilde{X}_{new}$ is selected using the equation below:

$$\tilde{X}_{new} = \tilde{X}_{old} + d \quad (3.1)$$

where d is a random number from a probability distribution specified by the user. Figure 3-1 shows the above described method of random walk sampling.

Figure 3-1: MCMC technique: Random Walk

Algorithm 1: The MCMC Algorithm

- 1: Draw a candidate design $\tilde{X}$ from the proposal distribution
 - 2: Evaluate the fitness function
 - 3: **if** fitness function is within user-specified range **then**
 - 4: accept $\tilde{X}$
 - 5: **else**
 - 6: accept the candidate design with a probability proportional to fitness function
 - 7: **end if**
 - 8: **repeat**
 - 9: Steps 1 to 7
 - 10: **until** user required number of solutions are obtained
-

Figure 3-2: Flowchart for the MCMC technique

3.2 Particle Swarm Optimization

PSO is an optimization technique inspired by the social behavior of bird flocks [52, 53]. Figure 3-3 shows a flock of birds and Fig. 3-4 shows a murmuration of starlings. It was introduced by Eberhart and Kennedy [29, 54]. This technique requires simple mathematical operators and is computationally inexpensive [29, 55].

Figure 3-3: A bird flock [56]

The algorithmic flow in PSO starts with a population of particles, whose positions represent the potential solutions for the studied problem. The particle velocities (rates of change of position of a particle) are randomly initialized in the search space. The search for the optimal position (solution) is performed by updating the particle velocities, hence positions, in each iteration/generation as follows. In every iteration, the fitness of each particles position is determined, and the velocity of each particle

Figure 3-4: A murmuration of starlings [57]

is updated by keeping track of the two “best” positions [58, 59, 60, 61]. The first is the best position a particle has traversed so far. This value is called p_{best} (particle). The second value is the best position that any particle has traversed so far. This best becomes the global best and is accordingly called g_{best} (global). Position and velocity are updated using the equations below.

$$\begin{aligned}\tilde{X}_{new} = \tilde{X}_{old} + c_1 \cdot rand \cdot (p_{best} - \tilde{P}_{old}) \\ + c_2 \cdot rand \cdot (g_{best} - \tilde{X}_{old})\end{aligned}\tag{3.2}$$

where $rand = \text{random number} \in [0 \ 1]$, $c_1 + c_2 = 4$ [62], and

$$\tilde{P}_{new} = \tilde{P}_{old} + \tilde{V}_{new}\tag{3.3}$$

Algorithm 2: The PSO Algorithm

```
1: Initialize the position and velocity of each particle between the bounds randomly
2: for each particle do
3:   Calculate fitness value and save  $p_{best}$  and  $g_{best}$ 
4:   Update velocity using (3.2)
5:   Update position using (3.3)
6:   Calculate fitness value
7:   if the current fitness value is better than  $p_{best}$  then
8:     Update  $p_{best}$ 
9:     Update  $g_{best}$  with the best fitness value among all particles
10:  end if
11: end for
12: Save  $g_{best}$  value
13: repeat
14:   Steps 1 to 12
15: until user required number of solutions are obtained
```

Figure 3-5: Flowchart for the PSO technique

3.3 Grey Wolf Optimizer

Grey Wolf optimizer is a recent algorithm proposed by S. Mirjalili et al. (2014) [47]. GWO was inspired by how the grey wolves hunt their prey. Grey wolves hunt their prey under the leadership of alpha wolf. The remaining wolves follow the orders of the alpha wolf. Sometimes alpha wolf takes the help of beta wolves, next level to alpha, in decision making. Then delta wolves and omega wolves follow. Delta wolves help beta wolves while omegas are considered as slaves. Omegas just follow the orders of all other wolves. Figure 3-6 shows how the grey wolves approach their prey and encircle it.

Figure 3-6: A bison is being chased by grey wolves. (A) Approach. (B, D) Encircling bison. (E) Grey wolves lying down [63].

The way grey wolves divide and allocate tasks resembles the way humans divide labor, which is a major feature of our society [64]. Like humans, who divide the complex processes (e.g. engineering manufacturing, or exploration in hostile environ-

ments, or a federal investigation) into simpler sub tasks, alpha grey wolves allocate tasks to those members of the pack who have proven ability and experience in completing them efficiently. In essence, the authors believe Grey Wolf Optimization is highly effective because it seems to emulate highly effective patterns consistent with human teams and their behaviors/strategies. From yet another perspective, the process of first encircling and then attacking the prey are akin to the expansion of the design space and its subsequent contraction.

S. Mirjalili et al. designed GWO algorithm such that the fittest solution is considered as the alpha (α) [47]. Accordingly, the second and third best solutions are named beta (β) and delta (δ) respectively. The remaining solutions are considered as omegas (ω). In the GWO algorithm, α , β , and δ wolves guide the hunting (optimization). The ω wolves follow these three wolves. *The order of dominance declines from alpha, beta, delta and omega in the same order.* This is visually represented in Fig. 3-7.

Figure 3-7: Hierarchy of grey wolves

The equations to update positions according to the Grey Wolf algorithm are as

follows:

$$\tilde{U} = 2 \cdot \tilde{w} \cdot \tilde{r}_1 - \tilde{w} \quad (3.4)$$

$$\tilde{V} = 2 \cdot \tilde{r}_2 \quad (3.5)$$

where elements in $\tilde{w}$ are linearly decreased from 2 to 0 over the period of iterations and $\tilde{r}_1, \tilde{r}_2$ are random variables uniformly distributed in $[0, 1]$, $\tilde{U}$ and $\tilde{V}$ are coefficient vectors.

$$\tilde{L}_i = |\tilde{V}_j \cdot \tilde{X}_i - \tilde{X}| \quad (3.6)$$

where $(i, j) = (\alpha, 1), (\beta, 2), (\delta, 3)$ and L is the distance between the prey and the grey wolf, X_i is the position of prey and X is the position of a grey wolf (candidate design).

$$\tilde{X}_m = \tilde{X}_n - \tilde{U}_m \cdot \tilde{L}_n \quad (3.7)$$

where $(m, n) = (1, \alpha), (2, \beta), (3, \delta)$

$$\tilde{X}(t+1) = \frac{\tilde{X}_1 + \tilde{X}_2 + \tilde{X}_3}{3} \quad (3.8)$$

Algorithm 3: The GWO Algorithm

- 1: Initialize the Grey Wolves, X_i ($i=1, 2, \dots, n$), $\tilde{w}$, $\tilde{U}$, & $\tilde{V}$
 - 2: Calculate the fitness of each candidate design (search agent) and save the first best solution as X_α , the second best solution as X_β , and the third best solution as X_δ
 - 3: **for** each candidate design (search agent) **do**
 - 4: **while** the conditions are satisfied **do**
 - 5: Update the position of the current candidate design (search agent) using (3.8)
 - 6: Update $\tilde{w}$, $\tilde{U}$, and $\tilde{V}$ using (3.4) and (3.5) and go to step 2
 - 7: Update X_α , X_β , X_δ
 - 8: **end while**
 - 9: **end for**
 - 10: Save values of X_α
 - 11: **repeat**
 - 12: Steps 1 to 10
 - 13: **until** user required number of solutions are obtained
-

Figure 3-8: Flowchart for the GWO technique

3.4 Artificial Neural Networks

Artificial neural networks are mathematical models motivated by the idea of replicating the functioning of the human brain. Theoretically, they are capable to learn and generalize any continuous multidimensional nonlinear input-output relationships [65, 66, 67]. The most important step in model development using neural networks is the neural network training. A neural network can be trained using a learning algorithm such as back propagation, conjugate gradient, and quasi-newton.

Figure 3-9: A typical neural network with multiple inputs and outputs

The weights and biases between the layers are assumed as follows:

- u_{ij} = weight between i^{th} input neuron and j^{th} hidden neuron

- v_{jk} = weight between j^{th} hidden neuron and k^{th} output neuron
- z_j = output of j^{th} hidden neuron
- p_j = bias to the j^{th} hidden neuron
- q_k = bias to the k^{th} output neuron

For a typical neural network in Fig. 3-9, with a input neurons (i), b hidden neurons (j) and c outputs (k), the equations for output from the neural network can be written as

$$\tilde{Y}_k = q_k + \sum_{j=1}^b v_{jk} \cdot z_j \quad (3.9)$$

where Y_k is the output from the k^{th} output layer neuron. A sigmoid function was chosen for hidden layer neuron output, which is

$$z_j = \frac{1}{1 + e^{-\gamma_j}} \quad (3.10)$$

$$\gamma_j = p_j + \sum_{i=1}^a u_{ij} \cdot X_i \quad (3.11)$$

$$\frac{\partial Y_k}{\partial X_i} = \sum_{j=1}^b v_{jk} \cdot z_j \cdot (1 - z_j) \cdot u_{ij} \quad (3.12)$$

$$X_{i(new)} = X_{i(old)} - \sum_{k=1}^c \frac{\partial Y_k}{\partial X_i} \quad (3.13)$$

The proposed method first calculates the derivative information ($\frac{\partial \tilde{Y}}{\partial X}$) from the forward model. Then the algorithm uses this derivative information to update the input value (candidate design) in each iteration using equation (3.13). The proposed

pseudocode of ANN-based method for an inverse problem consists of two parts as explained below:

Algorithm 4: The ANN Algorithm

PART 1

- 1: Get input and output data from the user (if a forward model already exists, then skip Step. 2)
- 2: Train a forward neural network model and calculate derivative information

PART 2

- 3: **for** each $\tilde{Y}_{given}$ **do**
 - 4: Initialize $\tilde{X}$ randomly between the boundaries
 - 5: Feed $\tilde{X}$ to the forward model to get $\tilde{Y}_{model}$ and calculate error
 - 6: **if** error is within user specified range **then**
 - 7: save the corresponding value of $\tilde{X}$
 - 8: **else**
 - 9: update $\tilde{X}$ using the derivative information from Step 2
 - 10: **end if**
 - 11: **end for**
 - 12: **repeat**
 - 13: Steps 1 to 11
 - 14: **until** user required number of solutions are obtained
-

Figure 3-10: Flowchart for the ANN technique

Chapter 4

Examples

This section demonstrates and compares the four methods on four engineering design problems involving a dynamic vibration absorber (DVA), a tension/compression spring, a pressure vessel design (PVD), and a ten-bar truss.

4.1 Dynamic Vibration Absorber

The objective is to find multiple designs of a DVA as illustrated in Fig. 4-2 [68]. A damped mass-spring system is subjected to force F . Resonance occurs if the forcing frequency is equal to the natural frequency ω_{n1} of the main mass. Resonance is a major issue for vibratory systems because it can cause failure. The forward model of this problem is:

Calculate system amplitude Y (normalized by the quasi-static response of mass M), given the design parameters ($M, m, \omega_{n1}, \omega_{n2}, \omega_{force}$). An inverse problem requires the calculation of the design parameters for a specified system amplitude. The mathematical formulation of this inverse problem is as follows:

Determine design parameters (output): m, ω_{n2}

So that the system amplitude (input):

$$0 \leq Y = |1 - (1/\beta_2)^2| / \{ [1 - R(1/\beta_1)^2 - (1/\beta_1)^2 - (1/\beta_2)^2 + 1/(\beta_1\beta_2)^2]^2 + 4\zeta^2[(1/\beta_1) - 1/(\beta_1\beta_2^2)]^2 \}^{0.5} \leq 20 \quad (4.1)$$

where $0.9 \leq \beta_1 = \frac{\omega_{n1}}{\omega_{force}}, \beta_2 = \frac{\omega_{n2}}{\omega_{force}} \leq 1.4$, $R = \frac{m}{M}$, and ζ = damping ratio of original system.

Figure 4-1: Cylindrical DVA attached to helicopter blades [69]

Figure 4-1 shows how DVAs are used in helicopters to reduce the vibrations of its wing blades. As mentioned earlier, the objective is to find many designs in terms of

$M, m, \omega_{n1}, \omega_{n2}, \omega_{force}$, so that the displacement of mass M satisfies equation (4.1). The boundaries for β_1 and β_2 are taken from F. Farizal et al. (2007) [68].

Figure 4-2: Tuned damper system

The performance of the techniques is determined by the elapsed time and error that is defined as follow:

$$Error = \sum_{i=1}^{200} |Y_{algorithm}(i) - Y_{given}(i)| \quad (4.2)$$

The solutions are accepted if and only if every $Error(i) \leq 0.01$. Therefore the error given in (4.2) should be less than 2. Error is normalized by the number of solutions

user requested, which is 200.

$$Error_{Normalized} = \frac{\sum_{i=1}^{200} |Y_{algorithm}(i) - Y_{given}(i)|}{200} \quad (4.3)$$

Table 4.1 compares the performance of the four techniques. The performance is evaluated based on the elapsed time and the error given by (4.2). The results show a consistent reduction in errors as well as in elapsed time from MCMC to GWO.

Table 4.1: Comparison of performance of the techniques for DVA

	MCMC	ANN		PSO	GWO
		SD*	SD + CG**		
Error (x10 ⁻⁴)	1036	1007.6	1020	978	900
Elapsed Time (s)	179.4	164.73	66	46.89	31.02
Normali- zed error (x10 ⁻⁵)	518	503.8	510	489	500

* Steepest Descent; **Conjugate Gradient

4.2 Tension Compression Spring

The objective of this problem is to find numerous designs of a tension/compression spring as illustrated in Fig. 4-3 [70, 71]. The spring is defined by three variables namely, wire diameter (d), mean coil diameter (D), and the number of active coils (N) [72]. The forward model of the problem involves the calculation of mass given the

design parameters. The inverse problem involves the calculation of design parameters for a specified mass. The mathematical formulation of this inverse problem is as follows:

Design parameters (output): $\tilde{X} = \{x_1, x_2, x_3\} = \{d, D, N\}$

Mass of the spring (input): $\tilde{Y} = \frac{\rho \cdot \pi \cdot x_1^2}{4} \cdot (\sqrt{1 + x_3 \cdot \pi \cdot x_2^2} + 2)$

Subject to $0.05 \leq x_1 \leq 2.00;$

$0.25 \leq x_2 \leq 1.30;$

$2.00 \leq x_3 \leq 15.0$

Figure 4-3: Tension/Compression Spring

Figure 4-4: Suspension spring in an automobile [73]

To measure the performance of the techniques, we use elapsed time and error as defined in (4.2):

The solutions are accepted if and only if every $\text{Error}(i) \leq 0.0001$. Therefore the error given in (4.2) should be less than 0.02. Error is normalized by number of solutions user requested, as defined in (4.3), which is 200.

Table 4.2 compares the performance of the four techniques. The performance is evaluated based on the elapsed time and the error given by (4.2). The results show a consistent reduction of both errors and elapsed time from MCMC to GWO. These observations are consistent with those in Table 4.1.

Table 4.2: Comparison of performance of the techniques for Tension/Compression Spring

	MCMC	ANN		PSO	GWO
		SD	SD + CG		
Error ($\times 10^{-4}$)	161	102	100	105	18
Elapsed Time (s)	43.23	24.27	10.77	6.43	3.68
Normali- zed error ($\times 10^{-5}$)	280	260	230	260	20.8

4.3 Pressure Vessel Design (PVD)

The objective of this problem is to find a design of PVDs as illustrated in Fig. 4-6. Figure 4-5 shows a typical pressure vessel. There are four variables in this problem: thickness of the shell (T_s), thickness of the head (T_h), inner radius (R), and length of the cylindrical section without considering the head (L) [74, 75]. The forward model of the problem involves the calculation of total cost consisting of material, forming, and welding of a cylindrical vessel given the design parameters. The inverse problem

involves the calculation of design parameters for a specified cost [75, 76].

Figure 4-5: Pressure Vessel [77]

Figure 4-6: Schematic of a Pressure Vessel [47]

The mathematical formulation of this inverse problem is as follows:

Design parameters (output): $\tilde{X} = \{x_1, x_2, x_3, x_4\} = \{T_s, T_h, R, L\}$

Cost function (input): $\tilde{Y} = 0.6224 \cdot x_1 \cdot x_3 \cdot x_4 + 1.7781 \cdot x_2 \cdot x_3^2 + 3.1661 \cdot x_1^2 \cdot x_4 + 19.84 \cdot x_1^2 \cdot x_3$

Subject to $0 \leq x_1 \leq 99;$

$0 \leq x_2 \leq 99$

$10 \leq x_3 \leq 200$

$10 \leq x_4 \leq 200$

To measure the performance of the techniques, we consider the elapsed time and the error in (4.2). The solutions are accepted if and only if every $\text{Error}(i) \leq 400$. Therefore the error given in (4.2) should be less than 80000. Error is normalized by number of solutions user requested, which is 200.

Table 4.3 compares the performance of the four techniques. The performance is evaluated based on the elapsed time and the error given by (4.2). The results show a consistent reduction of both error and elapsed time from MCMC to GWO. This

trend is also consistent with Tables 4.1 and 4.2.

Table 4.3: Comparison of performance of the techniques for Pressure Vessel design

	MCMC	ANN		PSO	GWO
		SD	SD + CG		
Error ($\times 10^{-4}$)	4.08	3.75	3.54	3.32	2.01
Elapsed Time (s)	363.95	184.23	130.75	105.40	63.82
Normali- zed error ($\times 10^{-5}$)	7.15	5.86	5.46	5.26	1.04

4.4 Ten Bar Truss Problem

R.T. Haftka et al. (1992) defined the TBT problem and discussed it in the context of inverse problem [78]. Figure 4-7 shows the Arawaka truss bridge in Japan. The ten-bar truss structure in Fig. 4-8 is under two loads, P_1 and P_2 . The objective is to find many designs of ten bar truss by changing the cross-sectional areas, A_i , of the truss members [79]. The design should fulfill stress constraints and minimum gage constraints. Input data are summarized in Table 4.4.

Figure 4-7: Arawaka truss bridge in Japan [80]

Table 4.4: Input summary

Parameters	Values
Dimension, b	360 in
Safety factor, S_f	1.5
Load, P_1	66.67 kips
Load, P_2	66.67 kips
Density, ρ	0.1 lb/in ³
Modulus of elasticity, E	10 ⁴ ksi
Allowable stress, σ	25 ksi
Allowable stress for member 9	75 ksi

Figure 4-8: Ten-Bar Truss with loads applied [78]

The forward model of the problem involves the calculation of safety factor given the design parameters [81]. The inverse problem involves the calculation of design parameters for a specified safety factor. The mathematical formulation of this inverse problem is as follows:

Design parameters (output): $\tilde{X} = \tilde{A}$

Safety factor (input): The equations to find safety factor are in [82]

Subject to $0.1 \leq A \leq 20$

To measure the performance of the techniques, we consider the elapsed time and error. The latter is defined as follows:

$$Error = \left| \frac{S_{fmin} - 1.5}{1.5} \right| \quad (4.4)$$

The solutions are accepted if and only if, $Error(i) \in [0.1, 3.5]$. The lower limit is specified for safety of the truss and the upper limit is specified considering economic limits. Therefore the error given above should be less than 700. Error is normalized by number of solutions that the user requested, given by (4.5), which is 200.

$$Error_{Normalized} = \frac{Error}{200} \quad (4.5)$$

Table 4.5 compares the performance of the four techniques. The performance is evaluated based on the elapsed time and the error given by (4.4). The results show a consistent reduction of both error and elapsed time from MCMC to GWO. This trend is also consistent with the trends observed in the previous tables.

Table 4.5: Comparison of performance of the techniques for the Ten-Bar Truss problem

	MCMC	ANN		PSO	GWO
		SD	SD + CG		
Error	27.72	25.85	25.13	24.73	23.84
Elapsed Time (s)	185.74	89.43	65.43	37.4	5.43
Normali- zed error ($\times 10^{-4}$)	1386	1292.5	1256.5	1236.5	1192

Chapter 5

Conclusions and Future Work

This thesis proposed a method and compared it with three methods for solving inverse problems. These inverse problems were formulated so that the performance of the four methods could be assessed on these problems. These were a novel ANN approach using simple backpropagation and conjugate gradient optimization for training the ANN, and GWO for solving the time-invariant inverse problems. The presented methods have benefits of compactness and ease of implementation. Grey Wolf optimizer, outperformed the other methods. This observation is consistent with that in S. Mirjalili et al. (2014) [47]. The authors believe Grey Wolf Optimization is very effective because it seeks to emulate this highly effective structure. For example, the processes of first encircling and then attacking the prey are akin to expansion of the design space and its subsequent contraction. The engineering examples presented in this thesis are proof-of-concept examples. They differ from industry based inverse problems by their complexity and design space. Since the four examples considered in this work cover a broad range of mechanical engineering aspects, it can be concluded that GWO's performance can be similar in every discipline.

While the results presented in this thesis demonstrated the effectiveness of the GWO and ANN techniques for solving inverse problems, further research can be done for the selection of best training method to reduce the error and computational time.

References

- [1] A. Tarantola, *Inverse problem theory and methods for model parameter estimation*. siam, 2005.
- [2] Indiamart.com, “Manufacturer of commercial spring,” 2015.
- [3] R. Snieder, “The role of nonlinearity in inverse problems,” *Inverse Problems*, vol. 14, no. 3, p. 387, 1998.
- [4] M. Brown, T. Fokas, Y. Kurylev, and B. Lionheart, “Inverse problems: Isaac newton institute for mathematical sciences,” 2015.
- [5] J. Sarvas, “Basic mathematical and electromagnetic concepts of the biomagnetic inverse problem,” *Physics in medicine and biology*, vol. 32, no. 1, p. 11, 1987.
- [6] R. L. Parker, “Inverse theory with grossly inadequate data,” *Geophysical Journal International*, vol. 29, no. 2, pp. 123–138, 1972.
- [7] A. I. Nachman, “Global uniqueness for a two-dimensional inverse boundary value problem,” *Annals of Mathematics*, pp. 71–96, 1996.
- [8] M. Bertero, C. De Mol, and E. R. Pike, “Linear inverse problems with discrete data. i. general formulation and singular system analysis,” *Inverse problems*, vol. 1, no. 4, p. 301, 1985.
- [9] M. Bertero and P. Boccacci, *Introduction to inverse problems in imaging*. CRC press, 1998.

- [10] A. I. Prilepko, D. G. Orlovsky, I. A. Vasin, *et al.*, *Methods for solving inverse problems in mathematical physics*. CRC Press, 2000.
- [11] M. Sambridge, “Geophysical inversion with a neighbourhood algorithm. searching a parameter space,” *Geophysical Journal International*, vol. 138, no. 2, pp. 479–494, 1999.
- [12] I. Flood and N. Kartam, “Neural networks in civil engineering. i: Principles and understanding,” *Journal of computing in civil engineering*, vol. 8, no. 2, pp. 131–148, 1994.
- [13] M. Sambridge and K. Mosegaard, “Monte carlo methods in geophysical inverse problems,” *Reviews of Geophysics*, vol. 40, no. 3, pp. 3–1, 2002.
- [14] G. Evensen and N. Fario, “Solving for the generalized inverse of the lorenz model,” *Journal-Meteorological Society Of Japan Series 2*, vol. 75, pp. 119–133, 1997.
- [15] Worldcat.org, “Worldcat.org: The world’s largest library catalog,” 2015.
- [16] Pet.med.wayne.edu, “Positron emission tomography (pet) center,” 2015.
- [17] D. Falkenberg, “Kit - welcome to institute of biomedical engineering,” 2015.
- [18] V. A. Ambartsumian, “Theoretical astrophysics.,” *New York, Pergamon Press [1958]*, vol. 1, 1958.
- [19] A. Tarantola and B. Valette, “Generalized nonlinear inverse problems solved using the least squares criterion,” *Rev. Geophys. Space Phys*, vol. 20, no. 2, pp. 219–232, 1982.
- [20] A. Tarantola and B. Valette, “Inverse problems= quest for information,” *J. geophys*, vol. 50, no. 3, pp. 150–170, 1982.

- [21] J.-P. Li, M. E. Balazs, G. T. Parks, and P. J. Clarkson, “A species conserving genetic algorithm for multimodal function optimization,” *Evolutionary computation*, vol. 10, no. 3, pp. 207–234, 2002.
- [22] K.-C. Wong, C.-H. Wu, R. K. Mok, C. Peng, and Z. Zhang, “Evolutionary multimodal optimization using the principle of locality,” *Information Sciences*, vol. 194, pp. 138–170, 2012.
- [23] S. Yazdani, H. Nezamabadi-pour, and S. Kamyab, “A gravitational search algorithm for multimodal optimization,” *Swarm and Evolutionary Computation*, vol. 14, pp. 1–14, 2014.
- [24] V. Keilis-Borok and T. Yanovskaja, “Inverse problems of seismology (structural review),” *Geophysical Journal International*, vol. 13, no. 1-3, pp. 223–234, 1967.
- [25] K. Mosegaard and M. Sambridge, “Monte carlo analysis of inverse problems,” *Inverse Problems*, vol. 18, no. 3, p. R29, 2002.
- [26] S. Geman and D. Geman, “Stochastic relaxation, gibbs distributions, and the bayesian restoration of images,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 721–741, 1984.
- [27] J. Marroquin, S. Mitter, and T. Poggio, “Probabilistic solution of ill-posed problems in computational vision,” *Journal of the american statistical association*, vol. 82, no. 397, pp. 76–89, 1987.
- [28] K. Mosegaard and A. Tarantola, “Monte carlo sampling of solutions to inverse problems,” *J. geophys. Res*, vol. 100, no. B7, pp. 12431–12447, 1995.
- [29] R. C. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1, pp. 39–43, New York, NY, 1995.

- [30] K. E. Parsopoulos, M. N. Vrahatis, *et al.*, “Particle swarm optimization method for constrained optimization problems,” *Intelligent Technologies—Theory and Application: New Trends in Intelligent Technologies*, vol. 76, pp. 214–220, 2002.
- [31] C. Mouser and S. Dunn, “Comparing genetic algorithms and particle swarm optimisation for an inverse problem exercise,” *ANZIAM Journal*, vol. 46, pp. 89–101, 2005.
- [32] J. Vesterstrøm and R. Thomsen, “A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems,” in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2, pp. 1980–1987, IEEE, 2004.
- [33] S. Ho, S. Yang, G. Ni, E. W. Lo, and H.-c. C. Wong, “A particle swarm optimization-based method for multiobjective design optimizations,” *Magnetics, IEEE Transactions on*, vol. 41, no. 5, pp. 1756–1759, 2005.
- [34] X. Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.
- [35] A. K. Jain, J. Mao, and K. Mohiuddin, “Artificial neural networks: A tutorial,” *Computer*, no. 3, pp. 31–44, 1996.
- [36] J. Khan, J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson, *et al.*, “Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks,” *Nature medicine*, vol. 7, no. 6, pp. 673–679, 2001.
- [37] H. A. Abbass, “An evolutionary artificial neural networks approach for breast cancer diagnosis,” *Artificial Intelligence in Medicine*, vol. 25, no. 3, pp. 265–281, 2002.

- [38] P. B. Snow, D. S. Smith, and W. J. Catalona, "Artificial neural networks in the diagnosis and prognosis of prostate cancer: a pilot study.," *The Journal of urology*, vol. 152, no. 5 Pt 2, pp. 1923–1926, 1994.
- [39] P. J. Lisboa and A. F. Taktak, "The use of artificial neural networks in decision support in cancer: a systematic review," *Neural networks*, vol. 19, no. 4, pp. 408–415, 2006.
- [40] K. J. Hunt, D. Sbarbaro, R. Żbikowski, and P. J. Gawthrop, "Neural networks for control systemsa survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.
- [41] B. Schölkopf, K.-K. Sung, C. J. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik, "Comparing support vector machines with gaussian kernels to radial basis function classifiers," *Signal Processing, IEEE Transactions on*, vol. 45, no. 11, pp. 2758–2765, 1997.
- [42] D. G. Fox, "Judging air quality model performance," *Bulletin of the American Meteorological Society*, vol. 62, no. 5, pp. 599–609, 1981.
- [43] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International journal of forecasting*, vol. 14, no. 1, pp. 35–62, 1998.
- [44] I. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," *Journal of microbiological methods*, vol. 43, no. 1, pp. 3–31, 2000.
- [45] S. R. H. Hoole, S. Subramaniam, R. Saldanha, J. Coulomb, and J. Sabonnadiere, "Inverse problem methodology and finite elements in the identification of cracks, sources, materials, and their geometry in inaccessible locations," *Magnetics, IEEE Transactions on*, vol. 27, no. 3, pp. 3433–3443, 1991.

- [46] S. R. H. Hoole, “Artificial neural networks in the solution of inverse electromagnetic field problems,” *Magnetics, IEEE Transactions on*, vol. 29, no. 2, pp. 1931–1934, 1993.
- [47] S. Mirjalili, S. M. Mirjalili, and A. Lewis, “Grey wolf optimizer,” *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014.
- [48] S. P. Brooks, “Markov chain monte carlo method and its application,” *The statistician*, pp. 69–100, 1998.
- [49] W. R. Gilks, *Markov chain monte carlo*. Wiley Online Library, 2005.
- [50] C. J. Geyer, “Practical markov chain monte carlo,” *Statistical Science*, pp. 473–483, 1992.
- [51] W. K. Hastings, “Monte carlo sampling methods using markov chains and their applications,” *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.
- [52] H. Qi, L.-M. Ruan, M. Shi, W. An, and H. Tan, “Application of multi-phase particle swarm optimization technique to inverse radiation problem,” *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 109, no. 3, pp. 476–493, 2008.
- [53] J. Kennedy, “Particle swarm optimization,” in *Encyclopedia of Machine Learning*, pp. 760–766, Springer, 2010.
- [54] J. Kennedy, “The particle swarm: social adaptation of knowledge,” in *Evolutionary Computation, 1997., IEEE International Conference on*, pp. 303–308, IEEE, 1997.
- [55] X. Wang, J. Yang, X. Teng, W. Xia, and R. Jensen, “Feature selection based on rough sets and particle swarm optimization,” *Pattern Recognition Letters*, vol. 28, no. 4, pp. 459–471, 2007.

- [56] Wonderopolis.org, “Do birds of a feather flock together?,” 2012.
- [57] Pacificnorthwestbirds.com, “Did you know what that flock of birds is actually called?,” 2015.
- [58] R. C. Eberhart and Y. Shi, “Particle swarm optimization: developments, applications and resources,” in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, pp. 81–86, IEEE, 2001.
- [59] Y. Shi and R. Eberhart, “A modified particle swarm optimizer,” in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pp. 69–73, IEEE, 1998.
- [60] H. Liu, Z. Cai, and Y. Wang, “Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization,” *Applied Soft Computing*, vol. 10, no. 2, pp. 629–640, 2010.
- [61] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, “Comprehensive learning particle swarm optimizer for global optimization of multimodal functions,” *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 3, pp. 281–295, 2006.
- [62] Y. Shi and R. C. Eberhart, “Parameter selection in particle swarm optimization,” in *Evolutionary programming VII*, pp. 591–600, Springer, 1998.
- [63] C. Muro, R. Escobedo, L. Spector, and R. Coppinger, “Wolf-pack (*canis lupus*) hunting strategies emerge from simple rules in computational simulations,” *Behavioural Processes*, vol. 88, no. 3, pp. 192 – 197, 2011.
- [64] A. Smith and A. Skinner, “The wealth of nations: books I–III,” 1982.

- [65] J. Ghaboussi, J. Garrett Jr, and X. Wu, “Knowledge-based modeling of material behavior with neural networks,” *Journal of Engineering Mechanics*, vol. 117, no. 1, pp. 132–153, 1991.
- [66] V. K. Devabhaktuni, *Neural Networks for RF and Microwave Design: Toward Automatic Model Generation*. PhD thesis, Carleton University Ottawa, 2003.
- [67] D. W. Patterson, *Artificial Neural Networks: Theory and Applications*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 1998.
- [68] F. Farizal and E. Nikolaidis, “Assessment of imprecise reliability using efficient probabilistic reanalysis,” tech. rep., SAE Technical Paper, 2007.
- [69] Oh1ninja.la.coocan.jp, “Oh-1 ninja scout helicopter homepage,” 2015.
- [70] J. W. Wittwer and L. L. Howell, “Design of a functionally binary pinned-pinned segment for use as a tension-compression spring in compliant micro mechanisms,” in *ASME 2002 International Mechanical Engineering Congress and Exposition*, pp. 197–204, American Society of Mechanical Engineers, 2002.
- [71] H. Blanc, G. Barrois, and C. Pisella, “Slider spring under” tension-compression”,” Mar. 22 1994. US Patent 5,296,983.
- [72] M. Mahdavi, M. Fesanghary, and E. Damangir, “An improved harmony search algorithm for solving optimization problems,” *Applied mathematics and computation*, vol. 188, no. 2, pp. 1567–1579, 2007.
- [73] Team-BHP.com, “Coil spring adjusters : Vfm fix for the honda civic’s (lousy) soft rear suspension? - team-bhp,” 2015.
- [74] H. H. Bednar, *Pressure Vessel Design Handbook*. Van Nostrand Reinhold, NY, 1985.

- [75] J. Spence and A. S. Tooth, “Pressure vessel design: concepts and principles,” 1994.
- [76] D. R. Moss and M. M. Basic, *Pressure vessel design manual*. Butterworth-Heinemann, 2012.
- [77] Fabconengg.com, “Pressure vessels — fabcon,” 2015.
- [78] R. T. Haftka and Z. Gürdal, *Elements of structural optimization*, vol. 11. Springer Science & Business Media, 1992.
- [79] W. Roux, N. Stander, and R. T. Haftka, “Response surface approximations for structural optimization,” *International Journal for Numerical Methods in Engineering*, vol. 42, no. 3, pp. 517–534, 1998.
- [80] En.yokogawa-bridge.co.jp, “Yokogawa bridge corp.,” 2015.
- [81] C. V. Camp and B. J. Bichon, “Design of space trusses using ant colony optimization,” *Journal of Structural Engineering*, vol. 130, no. 5, pp. 741–751, 2004.
- [82] S. Kumar, R. J. Pippy, E. Acar, N. H. Kim, and R. T. Haftka, “Approximate probabilistic optimization using exact-capacity-approximate-response-distribution (ecard),” *Structural and Multidisciplinary Optimization*, vol. 38, no. 6, pp. 613–626, 2009.

Appendix A

MATLAB code for MCMC

A.1 Dynamic Vibration Absorber

```
ns=input('Number of samples required: ');
R=0.01; zeta=0.01;
beta_upper_limit=1.4; beta_lower_limit=0.9;
A_given = load('A_given.txt'); % This is input file to the program
beta1_st=(beta_upper_limit-beta_lower_limit).*rand+beta_lower_limit;
beta2_st=(beta_upper_limit-beta_lower_limit).*rand+beta_lower_limit;
mud=0; % mean for d
sigmad=0.01; % sd for d
na=1;
while(na<=ns)
    d=normrnd(mud,sigmad,[1,2]); % increment for next point
    beta1_new=beta1_st + d(1,1); % Choosing next sample
    beta2_new=beta2_st + d(1,2); % Choosing next sample
    y_new = (abs(1-(1./beta2_new).^2))/((1-R.*(1./beta1_new).^2
        - (1./beta1_new).^2 - (1./beta2_new).^2 + (1./(beta1_new.
        *beta2_new)).^2 ).^2 + 4.*zeta.*zeta.*(1./beta1_new
```

```

        - (1./(beta1_new.*beta2_new.*beta2_new))).^2).^0.5;
alph = abs(A_given(na,1)-y_new);
if alph < 0.001 % Limits for the permissible y
if beta1_new>0.9 && beta1_new<1.4 && beta2_new>0.9 && beta2_new<1.4
beta_accept(na,1)= beta1_new; beta1_st = beta1_new;
beta_accept(na,2)= beta2_new; beta2_st = beta2_new;
ynew(na,1)=y_new;
na = na+1
end
end
end
for i=1:ns
y(i,1) =(abs(1-(1./beta_accept(i,2)).^2))/((1-R.*(1./beta_accept(i,1)).^2
- (1./beta_accept(i,1)).^2 - (1./beta_accept(i,2)).^2
+ (1./(beta_accept(i,1).*beta_accept(i,2))).^2 + 4.*zeta.*zeta
.*(1./beta_accept(i,1) - (1./(beta_accept(i,1).*beta_accept(i,2)
.*beta_accept(i,2)))).^2).^0.5;
end
E=0;
for i=1:ns
E = E + abs(y(i,1)-A_given(i,1));
end
En=0;
for i=1:ns
En = En + abs((y(i,1)-A_given(i,1))/y(i,1));
end
E_normalised = En/ns

```

A.2 Tension Compression Spring

```
ns=200;

ulim_x1=2.00;    %max value
llim_x1=0.05;    %min value
ulim_x2=1.30;    %max value
llim_x2=0.25;    %min value
ulim_x3=15;      %max value
llim_x3=2.00;    %min value

M_given=load('Mass.txt'); % input file to the program

x1 = round(100*(llim_x1+(ulim_x1-llim_x1)*rand))/100;
x2 = round(100*(llim_x2+(ulim_x2-llim_x2)*rand))/100;
x3 = randi([llim_x3 ulim_x3],1,1);

X_st(:,1) = x1;
X_st(:,2) = x2;
X_st(:,3) = x3;

mud=0; % mean for d
sigmad=0.1; % sd for d

na=1;

while(na<=ns)

%d=normrnd(mud,sigmad,[1,2]);

%increment for next point (random value from normal distribution)

%d(1,3)= randi([0 1],1,1);

% X_new = X_st + d; % Choosing next sample

x1 = round(100*(llim_x1+(ulim_x1-llim_x1)*rand))/100;
x2 = round(100*(llim_x2+(ulim_x2-llim_x2)*rand))/100;
x3 = randi([llim_x3 ulim_x3],1,1);
```

```

X_st(:,1) = x1; X_st(:,2) = x2; X_st(:,3) = x3; X_new = X_st;

rho = 7845/(1e6); % kg/m.^3

M_new = (rho.*pi.*(X_new(1,1).^2)/4).*(sqrt(1 + (X_new(1,3)
.*pi.*X_new(1,2)).^2) + 2);

alph = abs(M_given(na,1)-M_new);

if alph < 0.0001 % Limits for the permissible y
if X_new(1,1)>= 0.05 && X_new(1,1) <= 2.00 && X_new(1,2) >= 0.25
    && X_new(1,2) <= 1.30 && X_new(1,3)>= 2 && X_new(1,3)<= 15
X_accept(na,:)= X_new(1,:); X_st(1,:) = X_new(1,:); % accepting sample
Mnew(na,1)=M_new;

na = na+1

end

end

end

for i=1:ns
M(i,1) = (rho.*pi.*(X_accept(i,1).^2)/4).*(sqrt(1 + (X_accept(i,3)
.*pi.*X_accept(i,2)).^2) + 2);

end

E=0;

for i=1:ns
E = E + abs(M(i,1)-M_given(i,1));
end

En=0;

for i=1:ns
En = En + abs((M(i,1)-M_given(i,1))/M(i,1));
end

E_normalised = En/ns

```

A.3 Pressure Vessel Design

```
ns=200;

ulim_x1=99; %max value
llim_x1=0;   %min value
ulim_x2=99; %max value
llim_x2=0;   %min value
ulim_x3=200;%max value
llim_x3=10;  %min value
ulim_x4=200;%max value
llim_x4=10;  %min value

C_given=load('C_data.txt');

x1 = randi([llim_x1 ulim_x1],1,1);
x2 = randi([llim_x2 ulim_x2],1,1);
x3 = randi([llim_x3 ulim_x3],1,1);
x4 = randi([llim_x4 ulim_x4],1,1);

X_st(:,1) = x1;
X_st(:,2) = x2;
X_st(:,3) = x3;
X_st(:,4) = x4;

mud=0; % mean for d
sigmad=0.1; % sd for d

na=1;

while(na<=ns)

%d=normrnd(mud,sigmad,[1,2]);

% increment for next point (random value from normal distribution)

%d(1,3)= randi([0 1],1,1);
```

```

% X_new = X_st + d; % Choosing next sample
x1 = randi([llim_x1 ulim_x1],1,1);
x2 = randi([llim_x2 ulim_x2],1,1);
x3 = randi([llim_x3 ulim_x3],1,1);
x4 = randi([llim_x4 ulim_x4],1,1);
X_st(:,1) = x1;
X_st(:,2) = x2;
X_st(:,3) = x3;
X_st(:,4) = x4;
X_new = X_st;
C_new = 0.6224*(X_new(:,1)).*(X_new(:,3)).*(X_new(:,4)) + 1.7781
        *(X_new(:,2)).*(X_new(:,3).^2) + 3.1661*(X_new(:,1).^2)
        *(X_new(:,4)) + 19.84*(X_new(:,1).^2).*(X_new(:,3));
alph = abs(C_given(na,1)-C_new);
if alph < 400 % Limits for the permissible y
if X_new(1,1)>= 0 && X_new(1,1) <= 99 && X_new(1,2) >= 0 && X_new(1,2)
    <= 99 && X_new(1,3)>= 10 && X_new(1,3)<= 200 && X_new(1,4)>= 10
    && X_new(1,4)<= 200
X_accept(na,:)= X_new(1,:); %X_st(1,:) = X_new(1,:); % accepting sample
Cnew(na,1)=C_new;
na = na+1
end
end
end
for i=1:ns
C(i,1) = 0.6224*(X_accept(i,1)).*(X_accept(i,3)).*(X_accept(i,4))
        + 1.7781*(X_accept(i,2)).*(X_accept(i,3).^2) + 3.1661*

```

```

        (X_accept(i,1).^2).*(X_accept(i,4)) + 19.84*(X_accept(i,1).^2)
        .*(X_accept(i,3));
    end
    E=0;
    for i=1:ns
        E = E + abs(C(i,1)-C_given(i,1));
    end
    En=0;
    for i=1:ns
        En = En + abs((C(i,1)-C_given(i,1))/C(i,1));
    end
    E_normalised = En/ns

```

A.4 Ten Bar Truss Problem

```

P1 = 66.67;
P2=P1;
ns=101;
llim_A=0.1; %max value
ulim_A=10; %min value
mud=0;
sigmad=2;
na=1;
while(na<=ns)
    if na==1
        A = llim_A + (ulim_A-llim_A)*rand(10,1);
        a11 = (1/A(1) + 1/A(3) + 1/A(5) + 2*sqrt(2)/A(7) + 2*sqrt(2)/A(8));
    end
    na=na+1;
end

```

```

a12 = 1/A(5);
a21 = a12;
a22 = (1/A(2) + 1/A(4) + 1/A(5) + 1/A(6) + 2*sqrt(2)/A(9)
      + 2*sqrt(2)/A(10));
b1 = sqrt(2)*(P2/A(1) - (P1+2*P2)/A(3) - P2/A(5)
      - (P1+P2)*2*sqrt(2)/A(7));
b2 = -sqrt(2)*P2/A(4) - sqrt(2)*P2/A(5) - 4*P2/A(9);
N10 = (a11*b2 - a21*b1)./(a11*a22-a12*a21);
N9 = sqrt(2)*P2 + N10;
N8 = (b1*a22 - a12*b2)./(a11*a22 - a12*a21);
N7 = sqrt(2)*(P1+P2) + N8;
N6 = -N10./sqrt(2);
N5 = -P2 - N8./sqrt(2) - N10./sqrt(2);
N4 = -P2 - N10./sqrt(2);
N3 = -P1 -2*P2 -N8./sqrt(2);
N2 = -N10./sqrt(2);
N1 = P2 - N8./sqrt(2);
S = abs([N1/A(1);
N2/A(2);
N3/A(3);
N4/A(4);
N5/A(5);
N6/A(6);
N7/A(7);
N8/A(8);
N9/A(9);
N10/A(10);]);

```



```

SoF = 25./S;
SoF(9) = 75./S(9);
if min(SoF) >= 1.35 && max(SoF) <= 3.5
A_accept = A; % accepting sample
Stress_accept= S;
SOF_accept = SoF;
na = 2
end
else
d=normrnd(mud,sigmad,[10,1]);
A_new = A + d;
if min(A_new)>=0.1 && max(A_new)<=10
a11 = (1/A_new(1) + 1/A_new(3) + 1/A_new(5) + 2*sqrt(2)/A_new(7)
      + 2*sqrt(2)/A_new(8));
a12 = 1/A_new(5);
a21 = a12;
a22 = (1/A_new(2) + 1/A_new(4) + 1/A_new(5) + 1/A_new(6)
      + 2*sqrt(2)/A_new(9) + 2*sqrt(2)/A_new(10));
b1 = sqrt(2)*(P2/A_new(1) - (P1+2*P2)/A_new(3) - P2/A_new(5)
      - (P1+P2)*2*sqrt(2)/A_new(7));
b2 = -sqrt(2)*P2/A_new(4) - sqrt(2)*P2/A_new(5) - 4*P2/A_new(9);
N10 = (a11*b2 - a21*b1)./(a11*a22-a12*a21);
N9 = sqrt(2)*P2 + N10;
N8 = (b1*a22 - a12*b2)./(a11*a22 - a12*a21);
N7 = sqrt(2)*(P1+P2) + N8;
N6 = -N10./sqrt(2);
N5 = -P2 - N8./sqrt(2) - N10./sqrt(2);

```

```

N4 = -P2 - N10./sqrt(2);
N3 = -P1 -2*P2 -N8./sqrt(2);
N2 = -N10./sqrt(2);
N1 = P2 - N8./sqrt(2);
S = abs([N1/A_new(1);
N2/A_new(2);
N3/A_new(3);
N4/A_new(4);
N5/A_new(5);
N6/A_new(6);
N7/A_new(7);
N8/A_new(8);
N9/A_new(9);
N10/A_new(10);]);
SoF = 25./S;
SoF(9) = 75./S(9);
if min(SoF) >= 1.35 && max(SoF) <= 2.5
A_accept(:,na) = A; % accepting sample
Stress_accept(:,na)= S;
SOF_accept(:,na) = SoF;
na = na+1
A = A_new;
end
end
end
end

```

Appendix B

MATLAB code for PSO

B.1 Dynamic Vibration Absorber

```
ns = input('Enter number of solutions required: ');
np = 100; % no of particles
beta_upper_limit=1.4; beta_lower_limit=0.9;
A_given=load('A_given.txt'); R=0.01; zeta=0.01; c1=2; c2=2;
for i=1:ns
m=0; steps(i,1) = 0;
while(m==0)
beta_assume = (beta_upper_limit-beta_lower_limit).*rand(np,2)
    + beta_lower_limit;
vel = zeros(np,2) ;
for j = 1:np
A_pos(j,1) = (abs(1-(1./beta_assume(j,2)).^2))/((1-R.*(1./beta_assume
(j,1)).^2 - (1./beta_assume(j,1)).^2 - (1./beta_assume(j,2)).^2
+ (1./(beta_assume(j,1).*beta_assume(j,2))).^2 ).^2 + 4.*zeta.*zeta
.*(1./beta_assume(j,1) - (1./(beta_assume(j,1).*beta_assume(j,2)
.*beta_assume(j,2))))).^2).^0.5;
```

```

fitness(j,1) = abs(A_given(i,1) - (A_pos(j,1)));
if fitness(j,1)<= min(fitness)
pBest(j,1) = fitness(j,1);
else
pBest(j,1) = fitness(j,1);
end end

gBest = min(pBest);

for k=1:np
vel(k,:) = vel(k,:) +c1*rand*(pBest(k,1) - beta_assume(k,:))
+ c2*rand*(gBest - beta_assume(k,:)); steps(i,1) = steps(i,1) +1;
beta_assume(k,:) = beta_assume(k,:) + vel(k,:);
end

test = fitness<0.001;
for n=1:length(test)
if test(n)
beta_accept(i,:) = beta_assume(n,:); A_pos_accept(i,1) = A_pos(n,1);
m = 1;
end end end end

E=0;

for r=1:ns
E = E + abs(A_pos_accept(r,1)-A_given(r,1));
end

Error = E; E1=0;

for r=1:ns
E1 = E1 + abs((A_pos_accept(r,1)-A_given(r,1))/A_pos_accept(r,1));
end

Error_normalised = E1/ns

```

B.2 Tension Compression Spring

```
ns = 200; np = 100; % no of particles

ulim_x1=2.00;    %max value
llim_x1=0.05;    %min value
ulim_x2=1.30;    %max value
llim_x2=0.25;    %min value
ulim_x3=15;      %max value
llim_x3=2.00;    %min value

rho = 7845/(1e6); % kg/m.^3

M_given=load('Mass.txt'); c1=2; c2=2;

for i=1:ns
    m=0;
    steps(i,1) = 0;
    while(m==0)
        x1 = round(100*(llim_x1+(ulim_x1-llim_x1)*rand(np,1)))/100;
        x2 = round(100*(llim_x2+(ulim_x2-llim_x2)*rand(np,1)))/100;
        x3 = randi([llim_x3 ulim_x3],np,1);
        X(:,1) = x1; X(:,2) = x2; X(:,3) = x3; vel = zeros(np,3) ;
        for j = 1:np
            M_pos(j,1) = (rho.*pi.*(X(j,1).^2)/4).*(sqrt(1 + (X(j,3).*pi
                .*X(j,2)).^2) + 2);
            fitness(j,1) = abs(M_given(i,1) - (M_pos(j,1)));
            if fitness(j,1)<= min(fitness)
                pBest(j,1) = fitness(j,1);
            else
                pBest(j,1) = fitness(j,1);
            end
        end
    end
end
```

```

end end

gBest = min(pBest);

for k=1:np
vel(k,:) = vel(k,:) + c1.*rand.*(pBest(k,1) - X(k,:)) + c2*rand
    *(gBest - X(k,:));
steps(i,1) = steps(i,1) +1; X(k,:) = X(k,:) + vel(k,:);
end

test = fitness<0.0001;
for n=1:length(test)
if test(n)
X_accept(i,:) = X(n,:); M_pos_accept(i,1) = M_pos(n,1); m = 1;
end end end end E=0;

for r=1:ns
E = E + abs(M_pos_accept(r,1)-M_given(r,1));
end

Error = E; E1=0;

for r=1:ns
E1 = E1 + abs((M_pos_accept(r,1)-M_given(r,1))/M_pos_accept(r,1));
end

Error_normalised = E1/ns

```

B.3 Pressure Vessel Design

```

ns = 200;

np = 100; % no of particles

ulim_x1=99;    %max value

llim_x1=0;     %min value

```

```

ulim_x2=99;    %max value
llim_x2=0;    %min value
ulim_x3=200;    %max value
llim_x3=10;    %min value
ulim_x4=200;    %max value
llim_x4=10;    %min value
C_given=load('C_data.txt'); c1=2; c2=2;

for i=1:ns
m=0;
steps(i,1) = 0;
while(m==0)
x1 = randi([llim_x1 ulim_x1],np,1); x2 = randi([llim_x2 ulim_x2],np,1);
x3 = randi([llim_x3 ulim_x3],np,1); x4 = randi([llim_x4 ulim_x4],np,1);
X(:,1) = x1; X(:,2) = x2; X(:,3) = x3; X(:,4) = x4; X_new = X;
vel = zeros(np,4);
for j = 1:np
C_pso(j,1) = 0.6224*(X_new(j,1)).*(X_new(j,3)).*(X_new(j,4))
        + 1.7781*(X_new(j,2)).*(X_new(j,3).^2) + 3.1661*(X_new(j,1).^2)
        .*(X_new(j,4)) + 19.84*(X_new(j,1).^2).*(X_new(j,3));
fitness(j,1) = abs(C_given(i,1) - (C_pso(j,1)));
if fitness(j,1)<= min(fitness)
pBest(j,1) = fitness(j,1);
else
pBest(j,1) = fitness(j,1);
end end
gBest = min(pBest); test = fitness<400;
for n=1:length(test)

```

```

if test(n)
X_accept(i,:) = X(n,:); C_pso_accept(i,1) = C_pso(n,1); m = 1;
end end end end

E=0;

for r=1:ns
E = E + abs(C_pso_accept(r,1)-C_given(r,1));
end

Error = E; E1=0;

for r=1:ns
E1 = E1 + abs((C_pso_accept(r,1)-C_given(r,1))/C_pso_accept(r,1));
end

Error_normalised = E1/ns

```

B.4 Ten Bar Truss Problem

```

ns = 200; np =5; % no of particles

llim_A=0.1; %max value

ulim_A=20; %min value

P1 = 66.67; P2 = 66.67; c1=2; c2=2;

for i=1:ns
m=0;

while(m==0)
A = llim_A + (ulim_A-llim_A)*rand(10,np);

for j = 1:np
a11 = (1/A(1,j) + 1/A(3,j) + 1/A(5,j) + 2*sqrt(2)/A(7,j) + 2*sqrt(2)/
A(8,j));
a12 = 1/A(5,j); a21 = a12;

```



```

a22 = (1/A(2,j) + 1/A(4,j) + 1/A(5,j) + 1/A(6,j) + 2*sqrt(2)/A(9,j)
      + 2*sqrt(2)/A(10,j));
b1 = sqrt(2)*(P2/A(1,j) - (P1+2*P2)/A(3,j) - P2/A(5,j) - (P1+P2)*2
      *sqrt(2)/A(7,j));
b2 = -sqrt(2)*P2/A(4,j) - sqrt(2)*P2/A(5,j) - 4*P2/A(9,j);
N10 = (a11*b2 - a21*b1)./(a11*a22-a12*a21); N9 = sqrt(2)*P2 + N10;
N8 = (b1*a22 - a12*b2)./(a11*a22 - a12*a21);
N7 = sqrt(2)*(P1+P2) + N8; N6 = -N10./sqrt(2);
N5 = -P2 - N8./sqrt(2) - N10./sqrt(2); N4 = -P2 - N10./sqrt(2);
N3 = -P1 -2*P2 -N8./sqrt(2); N2 = -N10./sqrt(2); N1 = P2 - N8./sqrt(2);
S(:,j) = abs([N1/A(1,j); N2/A(2,j); N3/A(3,j); N4/A(4,j); N5/A(5,j);
N6/A(6,j); N7/A(7,j); N8/A(8,j); N9/A(9,j); N10/A(10,j)];);
end

SoF = 25./S; SoF(9,:) = 75./S(9,:); Error = abs((min(SoF) - 1.5)./1.5);
[error_min,index] = min(Error); Error_max = max(SoF(:,index));
if error_min <0.1 && Error_max < 3.5
A_accept(:,i) = A(:,index); Stress(:,i) = S(:,index);
SOF(:,i) = SoF(:,index);m=1;
end end end

```

Appendix C

MATLAB code for GWO

C.1 Dynamic Vibration Absorber

```
ns = 200; SearchAgents_no = 30; Max_iter = 2000; lb = 0.9; ub = 1.4;
dim = 2; R = 0.01; zeta = 0.01; A_given=load('A_given.txt');
for nsi=1:ns
    Alpha_pos=zeros(1,dim);
    Alpha_score=inf;
    Beta_pos=zeros(1,dim);
    Beta_score=inf;
    Delta_pos=zeros(1,dim);
    Delta_score=inf;
    Positions=initialization(SearchAgents_no,dim,ub,lb);
    l=0;% Loop counter
    while l<Max_iter
        for i=1:size(Positions,1)
            Flag4ub=Positions(i,:)>ub;
            Flag4lb=Positions(i,:)<lb;
            Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))
```

```

+ub.*Flag4ub+lb.*Flag4lb;

A_gwo = (abs(1-(1./Positions(i,2)).^2))/((1-R.*(1./Positions(i,1)).^2
- (1./Positions(i,1)).^2 - (1./Positions(i,2)).^2 + (1./(Positions(i,1)
.*Positions(i,2))).^2 ).^2 + 4.*zeta.*zeta.*(1./Positions(i,1)
- (1./(Positions(i,1).*Positions(i,2).*Positions(i,2))))).^2).^0.5;
fitness= abs(A_given(nsi,1) - A_gwo);
if fitness<Alpha_score
    Alpha_score=fitness; % Update alpha
    Alpha_pos=Positions(i,:);
end
if fitness>Alpha_score && fitness<Beta_score
    Beta_score=fitness; % Update beta
    Beta_pos=Positions(i,:);
end
if fitness>Alpha_score && fitness>Beta_score
    && fitness<Delta_score
        Delta_score=fitness; % Update delta
        Delta_pos=Positions(i,:);
    end end
a=2-1*((2)/Max_iter); % a decreases linearly from 2 to 0
for i=1:size(Positions,1)
    for j=1:size(Positions,2)
        r1=rand(); % r1 is a random number in [0,1]
        r2=rand(); % r2 is a random number in [0,1]
        A1=2*a*r1-a; % Equation (3.3)
        C1=2*r2; % Equation (3.4)
        D_alpha=abs(C1*Alpha_pos(j)-Positions(i,j));

```

```

X1=Alpha_pos(j)-A1*D_alpha;
r1=rand();          r2=rand();
A2=2*a*r1-a; % Equation (3.3)
C2=2*r2; % Equation (3.4)
D_beta=abs(C2*Beta_pos(j)-Positions(i,j));
X2=Beta_pos(j)-A2*D_beta; % Equation (3.6)-part 2
r1=rand();          r2=rand();
A3=2*a*r1-a; % Equation (3.3)
C3=2*r2; % Equation (3.4)
D_delta=abs(C3*Delta_pos(j)-Positions(i,j));
X3=Delta_pos(j)-A3*D_delta; % Equation (3.5)-part 3
Positions(i,j)=(X1+X2+X3)/3;% Equation (3.7)
end end

l=l+1;
end
Error(nsi,1) = Alpha_score; beta_accept(nsi,:) = Alpha_pos;
end
for i=1:ns
A_gwo_accept(i,1) = (abs(1-(1./beta_accept(i,2)).^2))/((1-R
.*(1./beta_accept(i,1)).^2 - (1./beta_accept(i,1)).^2
- (1./beta_accept(i,2)).^2 + (1./(beta_accept(i,1)
.*beta_accept(i,2))).^2 ).^2 + 4.*zeta.*zeta.*(1./beta_accept(i,1)
- (1./(beta_accept(i,1).*beta_accept(i,2).*beta_accept(i,2))))).^2).^0.5;
end
E=0;
for r=1:ns
E = E + abs(A_gwo_accept(r,1)-A_given(r,1));

```

```

end

Error_gwo = E

E1=0;

for s=1:ns

E1 = E1 + abs((A_gwo_accept(s,1)-A_given(s,1))/A_gwo_accept(s,1));

end

Error_normalised = E1/ns

```

C.2 Tension Compression Spring

```

ns = 200; %input('Enter number of solutions required: ');
SearchAgents_no = 5;
Max_iter = 200; %input('Enter max number of iterations: ');
ub = [2.00 1.30 15];
lb = [0.05 0.25 2.00];
rho = 7845/(1e6); % kg/m.^3
dim = 3;
M_given=load('Mass.txt');
for nsi=1:ns

Alpha_pos=zeros(1,dim);
Alpha_score=inf;

Beta_pos=zeros(1,dim);
Beta_score=inf;

```

```

Delta_pos=zeros(1,dim);
Delta_score=inf;

%Initialize the positions of search agents
Positions=initialization(SearchAgents_no,dim,ub,lb);

l=0;% Loop counter

while l<Max_iter
for i=1:size(Positions,1)

Flag4ub=Positions(i,:)>ub;
Flag4lb=Positions(i,:)<lb;
Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb
                .*Flag4lb;

% Calculate objective function for each search agent
M_gwo = (rho.*pi.*(Positions(i,1).^2)/4).*(sqrt(1 + (Positions(i,3)
                .*pi.*Positions(i,2)).^2) + 2);

fitness = abs(M_given(nsi,1) - M_gwo);

% Update Alpha, Beta, and Delta
if fitness<Alpha_score
Alpha_score=fitness; % Update alpha
Alpha_pos=Positions(i,:);
end

```

```

if fitness>Alpha_score && fitness<Beta_score
Beta_score=fitness; % Update beta
Beta_pos=Positions(i,:);
end

if fitness>Alpha_score && fitness>Beta_score && fitness<Delta_score
Delta_score=fitness; % Update delta
Delta_pos=Positions(i,:);
end

end

a=2-1*((2)/Max_iter); % a decreases linearly from 2 to 0
% Update the Position of search agents including omegas
for i=1:size(Positions,1)
for j=1:size(Positions,2)
r1=rand(); % r1 is a random number in [0,1]
r2=rand(); % r2 is a random number in [0,1]
A1=2*a*r1-a; % Equation (3.3)
C1=2*r2; % Equation (3.4)
D_alpha=abs(C1*Alpha_pos(j)-Positions(i,j)); % Equation (3.5)-part 1
X1=Alpha_pos(j)-A1*D_alpha; % Equation (3.6)-part 1
r1=rand();
r2=rand();
A2=2*a*r1-a;
C2=2*r2;
D_beta=abs(C2*Beta_pos(j)-Positions(i,j));
X2=Beta_pos(j)-A2*D_beta;
r1=rand();
r2=rand();

```

```

A3=2*a*r1-a;
C3=2*r2;
D_delta=abs(C3*Delta_pos(j)-Positions(i,j));
X3=Delta_pos(j)-A3*D_delta;
Positions(i,j)=(X1+X2+X3)/3;
end end

l=l+1;

end

Error(nsi,1) = Alpha_score;
X_accept(nsi,:) = Alpha_pos;
end

for i=1:ns
M_gwo_accept(i,1) = (rho.*pi.*(X_accept(i,1).^2)/4).*(sqrt(1 + (X_accept(i,3).*pi.
end

E=0;

for r=1:ns
E = E + abs(M_gwo_accept(r,1)-M_given(r,1));
end

Error_gwo = E

E1=0;

for s=1:ns
E1 = E1 + abs((M_gwo_accept(s,1)-M_given(s,1))/M_gwo_accept(s,1));
end

Error_normalised = E1/ns

```


C.3 Pressure Vessel Design

```
ns = 200; %input('Enter number of solutions required: ');
SearchAgents_no = 20;
Max_iter = 800;
ub = [99 99 200 200];
lb = [0 0 10 10];
dim = 4;
C_given=load('\C_data.txt');

for nsi=1:ns
    Alpha_pos=zeros(1,dim);
    Alpha_score=inf;

    Beta_pos=zeros(1,dim);
    Beta_score=inf;

    Delta_pos=zeros(1,dim);
    Delta_score=inf;

    Positions=initialization(SearchAgents_no,dim,ub,lb);
    l=0;% Loop counter
    while l<Max_iter
        for i=1:size(Positions,1)
            Flag4ub=Positions(i,:)>ub;
            Flag4lb=Positions(i,:)<lb;
            Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+ub
```

```

.*Flag4ub+lb.*Flag4lb;

C_gwo = 0.6224*(Positions(i,1)).*(Positions(i,3)).*(Positions(i,4))
      + 1.7781*(Positions(i,2)).*(Positions(i,3).^2)
      + 3.1661*(Positions(i,1).^2).*(Positions(i,4))
      + 19.84*(Positions(i,1).^2).*(Positions(i,3));
fitness = abs(C_given(nsi,1) - C_gwo);
% Update Alpha, Beta, and Delta
if fitness<Alpha_score
Alpha_score=fitness; % Update alpha
Alpha_pos=Positions(i,:);
end
if fitness>Alpha_score && fitness<Beta_score
Beta_score=fitness; % Update beta
Beta_pos=Positions(i,:);
end
if fitness>Alpha_score && fitness>Beta_score && fitness<Delta_score
Delta_score=fitness; % Update delta
Delta_pos=Positions(i,:);
end end
a=2-1*((2)/Max_iter);
for i=1:size(Positions,1)
for j=1:size(Positions,2)
r1=rand(); % r1 is a random number in [0,1]
r2=rand(); % r2 is a random number in [0,1]
A1=2*a*r1-a;
C1=2*r2;

```

```

D_alpha=abs(C1*Alpha_pos(j)-Positions(i,j));
X1=Alpha_pos(j)-A1*D_alpha;
r1=rand();
r2=rand();
A2=2*a*r1-a;
C2=2*r2;
D_beta=abs(C2*Beta_pos(j)-Positions(i,j));
X2=Beta_pos(j)-A2*D_beta;
r1=rand();
r2=rand();
A3=2*a*r1-a;
C3=2*r2;
D_delta=abs(C3*Delta_pos(j)-Positions(i,j));
X3=Delta_pos(j)-A3*D_delta;
Positions(i,j)=(X1+X2+X3)/3;
end end

l=l+1;

end

Error(nsi,1) = Alpha_score;
X_accept(nsi,:) = Alpha_pos;
end

for i=1:ns
C_gwo_accept(i,1) = 0.6224*(X_accept(i,1)).*(X_accept(i,3))
                .*(X_accept(i,4)) + 1.7781*(X_accept(i,2)).*(X_accept(i,3).^2)
                + 3.1661*(X_accept(i,1).^2).*(X_accept(i,4))
                + 19.84*(X_accept(i,1).^2).*(X_accept(i,3));
end

```

```

E=0;

for r=1:ns
E = E + abs(C_gwo_accept(r,1)-C_given(r,1));
end

Error_gwo = E

E1=0;

for s=1:ns
E1 = E1 + abs((C_gwo_accept(s,1)-C_given(s,1))/C_gwo_accept(s,1));
end

Error_normalised = E1/ns

```

C.4 Ten Bar Truss Problem

```

ns = 100; %input('Enter number of solutions required: ');
SearchAgents_no = 20;
Max_iter = 20;
ub = 10; lb = 0.1;
P1 = 66.67; P2 = 66.67;
dim = 10;
for nsi=1:ns
q=0;
while(q==0)

Alpha_pos=zeros(1,dim);
Alpha_score=inf;

Beta_pos=zeros(1,dim);

```

```

Beta_score=inf;

Delta_pos=zeros(1,dim);
Delta_score=inf;

Positions=initialization(SearchAgents_no,dim,ub,lb);
l=0;% Loop counter
while l<Max_iter
for i=1:size(Positions,1)
Flag4ub=Positions(i,:)>ub;
Flag4lb=Positions(i,:)<lb;
Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb
                .*Flag4lb;
% Calculate objective function for each search agent
S(i,:)=Stress(Positions(i,:),P1,P2);
SoF(i,:) = 25./S(i,:);
SoF(i,9) = 75./S(i,9);
fitness = max([abs(max(SoF(i,:))-1.925) abs(min(SoF(i,:))-1.925)]);
% Update Alpha, Beta, and Delta
if fitness<Alpha_score
Alpha_score=fitness; % Update alpha
Alpha_pos=Positions(i,:);
end
if fitness>Alpha_score && fitness<Beta_score
Beta_score=fitness; % Update beta
Beta_pos=Positions(i,:);
end

```

```

if fitness>Alpha_score && fitness>Beta_score && fitness<Delta_score
Delta_score=fitness; % Update delta
Delta_pos=Positions(i,:);
end end

a=2-1*((2)/Max_iter);
for i=1:size(Positions,1)
for j=1:size(Positions,2)
r1=rand(); % r1 is a random number in [0,1]
r2=rand(); % r2 is a random number in [0,1]
A1=2*a*r1-a;
C1=2*r2;
D_alpha=abs(C1*Alpha_pos(j)-Positions(i,j));
X1=Alpha_pos(j)-A1*D_alpha;
r1=rand();
r2=rand();
A2=2*a*r1-a;
C2=2*r2;
D_beta=abs(C2*Beta_pos(j)-Positions(i,j));
X2=Beta_pos(j)-A2*D_beta;
r1=rand();
r2=rand();
A3=2*a*r1-a;
C3=2*r2;
D_delta=abs(C3*Delta_pos(j)-Positions(i,j));
X3=Delta_pos(j)-A3*D_delta;
Positions(i,j)=(X1+X2+X3)/3;
end end

```

```

l=l+1;
end
checkstress = Stress(Alpha_pos,P1,P2);
checkFoS = 25./checkstress;
checkFoS(:,9) = 75./checkstress(:,9);
maxcheck=max(checkFoS);
mincheck=min(checkFoS);
if maxcheck <=2.5 && mincheck >= 1.35
A_accept(nsi,:)=Alpha_pos;
q=1;
end end end

for i=1:size(A_accept,1)
finalstress(i,:) = Stress(A_accept(i,:),P1,P2);
end

```

Appendix D

MATLAB code for ANN

D.1 Dynamic Vibration Absorber

```
mincb=0.001; %min beta update value
maxcb=0.5;% ma3x beta update value
ratiomin=mincb/maxcb; %lower limit for ratio
ratiomax=maxcb/mincb; %upper limit for ratio
ns = input('Enter number of solutions required: ');
beta_upper_limit=1.4;
beta_lower_limit=0.9;
A_given=load('A_given.txt');
iterations = load('iterations.txt');
u = transpose(load('weights_u.dat'));
v = load('weights_v.dat');
p = load('weights_p.dat');
q = load('weights_q.dat');
R=0.01;
zeta=0.01;
tic
```



```

for i = 1:ns
r=1;
s=0;
itr(i,1)=0;
g2 = 0;
iter(i,1) = 0;
if itr(i,1) <= (iterations(i,1)/2)
while(g2 == 0)
beta_assume = (beta_upper_limit-beta_lower_limit).*rand(1,2)
+ beta_lower_limit;
k=0;
while(k == 0)
A_forward_model = (abs(1-(1./beta_assume(1,2)).^2))/((1-R
.*(1./beta_assume(1,1)).^2 - (1./beta_assume(1,1)).^2
- (1./beta_assume(1,2)).^2 + (1./(beta_assume(1,1)
.*beta_assume(1,2))).^2 ).^2 + 4.*zeta.*zeta.*(1./beta_assume(1,1)
- (1./(beta_assume(1,1).*beta_assume(1,2)
.*beta_assume(1,2)))).^2).^0.5;
z = z_value(beta_assume);
alph = abs(A_given(i,1) - A_forward_model);

if alph < 0.001 % accepting close amplitudes
beta_accept(i,:) = beta_assume;
A_from_ann(i,:) = A_forward_model;
k=1;
g2=1;
else

```

```

% Derivatives
dA_db1 = 0;
dA_db2 = 0;
for l = 1:size(u,2)
dA_db1 = dA_db1 + v(l,1)*z(l,1)*(1-z(l,1))*u(1,l);
dA_db2 = dA_db2 + v(l,1)*z(l,1)*(1-z(l,1))*u(2,l);
end
m= dA_db1/abs(dA_db1);
n= dA_db2/abs(dA_db2);
ratio=abs(dA_db1/dA_db2);
if ratio <=ratiomin
dA_db2 = n*maxcb;
dA_db1=0;
elseif ratio>=ratiomax
dA_db1=m*maxcb;
dA_db2=0;
elseif ratio<1
dA_db2=n*maxcb;
dA_db1=m*ratio*maxcb;
else
dA_db2=n*mincb;
dA_db1=m*ratio*mincb;
end

dA_dbeta(:,1) = dA_db1;
dA_dbeta(:,2) = dA_db2;
beta_assume=beta_assume - dA_dbeta;

```

```

iter(i,1) = iter(i,1)+1;
itr(i,1) = itr(i,1) +1;
if beta_assume(1)<0.9 || beta_assume(1)>1.4||beta_assume(2)<0.9
    || beta_assume(2)>1.4 || alph > 0.8
k=1;
iter(i,1) = 0;
itr(i,1)=0;
end end end end
iter(i,1) = 0;
g1 = 0;
while(g1 == 0)
k=0;
while(k == 0)
A_forward_model = (abs(1-(1./beta_assume(1,2)).^2))/((1-R
    .*(1./beta_assume(1,1)).^2 - (1./beta_assume(1,1)).^2
    - (1./beta_assume(1,2)).^2 + (1./(beta_assume(1,1)
    .*beta_assume(1,2))).^2 ).^2 + 4.*zeta.*zeta
    .*(1./beta_assume(1,1) - (1./(beta_assume(1,1).*beta_assume(1,2)
    .*beta_assume(1,2))))).^2).^0.5;
z = z_value(beta_assume);
alph = abs(A_given(i,1) - A_forward_model);

if alph < 0.001 % accepting close amplitudes
beta_accept(i,:) = beta_assume;
A_from_ann(i,:) = A_forward_model;
k=1;
g1=1;

```

```

else
% Derivatives
dA_db1 = 0;
dA_db2 = 0;
for l = 1:size(u,2)
dA_db1 = dA_db1 + v(l,1)*z(l,1)*(1-z(l,1))*u(1,l);
dA_db2 = dA_db2 + v(l,1)*z(l,1)*(1-z(l,1))*u(2,l);
end
m= dA_db1/abs(dA_db1);
n= dA_db2/abs(dA_db2);
ratio=abs(dA_db1/dA_db2);
if ratio <=ratiomin
dA_db2 = n*maxcb;
dA_db1=0;
elseif ratio>=ratiomax
dA_db1=m*maxcb;
dA_db2=0;
elseif ratio<1
dA_db2=n*maxcb;
dA_db1=m*ratio*maxcb;
else
dA_db2=n*mincb;
dA_db1=m*ratio*mincb;
end

dA_dbeta(:,1) = dA_db1;
dA_dbeta(:,2) = dA_db2;

```

```

if s==0
g(r,:) = dA_dbeta;
h(r,:) = -g(r,:) ;
beta_assume = beta_assume + 0.1*h(r,:);
s=1;
r=r+1;
else
g(r,:) = dA_dbeta;
gamma(r-1,:) = (g(r,:)*g(r,:))'./(g(r-1,:)*g(r-1,:))';
h(r,:) = - g(r,:) + gamma(r-1,:)*h(r-1,:);
beta_assume = beta_assume + 0.1*h(r,:);
r = r+1;
end
iter(i,1) = iter(i,1)+1;
if beta_assume(1)<0.9 || beta_assume(1)>1.4||beta_assume(2)<0.9
    || beta_assume(2)>1.4 || alph > 0.8
k=1;
r=2;
iter(i,1) = 0;
end end end end end end
for i=1:ns
y(i,1) = (abs(1-(1./beta_accept(i,2)).^2))/((1-R
    .*(1./beta_accept(i,1)).^2 - (1./beta_accept(i,1)).^2
    - (1./beta_accept(i,2)).^2 + (1./(beta_accept(i,1)
    .*beta_accept(i,2))).^2 ).^2 + 4.*zeta.*zeta.*(1
    ./beta_accept(i,1) - (1./(beta_accept(i,1).*beta_accept(i,2)

```

```

        .*beta_accept(i,2))))).^2).^0.5;
end
E=0;
for i=1:ns
E = E + abs(y(i,1)-A_given(i,1));
end
Error = E;
E1=0;
for i=1:ns
E1 = E1 + abs((y(i,1)-A_given(i,1))/y(i,1));;
end
Error_normalised = E1/ns

```

D.2 Tension Compression Spring

```

minc=[0.01 0.01 1]; %min beta update value
maxc=[1.9 1 12];% max beta update value
ratiomin=minc./maxc; %lower limit for ratio
ratiomax=maxc./minc; %upper limit for ratio
ns = 200;

ulim_x1=2.00;    %max value
llim_x1=0.05;    %min value
ulim_x2=1.30;    %max value
llim_x2=0.25;    %min value
ulim_x3=15;      %max value
llim_x3=2.00;    %min value
rho = 7845/(1e6); % kg/m.^3

```

```

iterations=load('iterations.txt');
M_given=load('Mass.txt');
u = transpose(load('weights_u.dat'));
v = load('weights_v.dat');
p = load('weights_p.dat');
q = load('weights_q.dat');
for i = 1:ns
r=1;
s=0;
itr(i,1)=0;
g2 = 0;
if itr(i,1) <= (iterations(i,1)/2)
while(g2 == 0)
x1 = round(100*(llim_x1+(ulim_x1-llim_x1)*rand))/100;
x2 = round(100*(llim_x2+(ulim_x2-llim_x2)*rand))/100;
x3 = randi([llim_x3 ulim_x3],1,1);
X_assume(:,1) = x1;
X_assume(:,2) = x2;
X_assume(:,3) = x3;
k=0;
while(k == 0)
M_forward_model = (rho.*pi.*(X_assume(:,1).^2)/4).*(sqrt(1
+ (X_assume(:,3).*pi.*X_assume(:,2)).^2) + 2);
z = z_value(X_assume);
alph = abs(M_given(i,1) - M_forward_model);

if alph < 0.0001    % accepting close amplitudes

```

```

X_accept(i,:) = X_assume;
M_from_ann(i,:) = M_forward_model;

k=1;

g=1;

g2=1;

else

% Derivatives

dM_dx1 = 0;

dM_dx2 = 0;

dM_dx3 = 0;

for l = 1:size(u,2)

dM_dx1 = dM_dx1 + v(l,1)*z(l,1)*(1-z(l,1))*u(1,l);

dM_dx2 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(2,l);

dM_dx3 = dM_dx3 + v(l,1)*z(l,1)*(1-z(l,1))*u(3,l);

end

dM_dX(:,1) = dM_dx1;

dM_dX(:,2) = dM_dx2;

dM_dX(:,3) = dM_dx3;

X_assume = X_assume - 0.1*dM_dX;

X_assume(:,3) = round(X_assume(:,3));

itr(i,1) = itr(i,1) +1;

if X_assume(1)<0.05 || X_assume(1)>2.00||X_assume(2)<0.25

    || X_assume(2)>1.3 || X_assume(3)<2 || X_assume(3)>15 || alph > 0.05

k=1;

itr(i,1)=0;

end end end end

g1 = 0;

```



```

while(g1 == 0)
k=0;
while(k == 0)
% M_forward_model = forward(X_assume);
M_forward_model = (rho.*pi.*(X_assume(:,1).^2)/4).*(sqrt(1
    + (X_assume(:,3).*pi.*X_assume(:,2)).^2) + 2);
z = z_value(X_assume);
alph = abs(M_given(i,1) - M_forward_model);
if alph < 0.0001 % accepting close amplitudes
X_accept(i,:) = X_assume;
M_from_ann(i,:) = M_forward_model;
k=1;
g1=1;
else
% Derivatives
dM_dx1 = 0;
dM_dx2 = 0;
dM_dx3 = 0;
for l = 1:size(u,2)
dM_dx1 = dM_dx1 + v(l,1)*z(l,1)*(1-z(l,1))*u(1,l);
dM_dx2 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(2,l);
dM_dx3 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(3,l);
end
dM_dX(:,1) = dM_dx1;
dM_dX(:,2) = dM_dx2;
dM_dX(:,3) = dM_dx3;
if s==0

```

```

g(r,:) = dA_dX;
h(r,:) = -g(r,:) ;
X_assume = X_assume + 0.1*h(r,:);
s=1;
r=r+1;
else
g(r,:) = dA_dX;
gamma(r-1,:) = (g(r,:)*g(r,:))'./(g(r-1,:)*g(r-1,:))';
h(r,:) = - g(r,:) + gamma(r-1,:)*h(r-1,:);
X_assume = X_assume + 0.1*h(r,:);
r = r+1;
end
if X_assume(1)<0.05 || X_assume(1)>2.00||X_assume(2)<0.25
    || X_assume(2)>1.3 || X_assume(3)<2 || X_assume(3)>15 || alph > 2
k=1;
r=2;
end end end end end end
for i=1:ns
M(i,1) = (rho.*pi.*(X_accept(i,1).^2)/4).*(sqrt(1
    + (X_accept(i,3).*pi.*X_accept(i,2)).^2) + 2);
end
E=0;
for i=1:ns
E = E + abs(M(i,1)-M_given(i,1));
end
Error = E
E1=0;

```

```

for i=1:ns
E1 = E1 + abs((M(i,1)-M_given(i,1))/M(i,1));
end
Error_normalised = E1/ns

```

D.3 Pressure Vessel Design

```

minc=[0.01 0.01 1]; %min beta update value
maxc=[1.9 1 12]; % max beta update value
ratiomin=minc./maxc; %lower limit for ratio
ratiomax=maxc./minc; %upper limit for ratio
ns = 200;

ulim_x1=2.00; %max value
llim_x1=0.05; %min value
ulim_x2=1.30; %max value
llim_x2=0.25; %min value
ulim_x3=15; %max value
llim_x3=2.00; %min value
rho = 7845/(1e6); % kg/m.^3
iterations=load('iterations.txt');
M_given=load('Mass.txt');
u = transpose(load('weights_u.dat'));
v = load('weights_v.dat');
p = load('weights_p.dat');
q = load('weights_q.dat');
for i = 1:ns

```

```

r=1;
s=0;
itr(i,1)=0;
g2 = 0;
if itr(i,1) <= (iterations(i,1)/2)
while(g2 == 0)
x1 = round(100*(llim_x1+(ulim_x1-llim_x1)*rand))/100;
x2 = round(100*(llim_x2+(ulim_x2-llim_x2)*rand))/100;
x3 = randi([llim_x3 ulim_x3],1,1);
X_assume(:,1) = x1;
X_assume(:,2) = x2;
X_assume(:,3) = x3;
k=0;
while(k == 0)
% M_forward_model = forward(X_assume);
M_forward_model = (rho.*pi.*(X_assume(:,1).^2)/4).*(sqrt(1
    + (X_assume(:,3).*pi.*X_assume(:,2)).^2) + 2);
z = z_value(X_assume);
alph = abs(M_given(i,1) - M_forward_model);
if alph < 0.0001    % accepting close amplitudes
X_accept(i,:) = X_assume;
M_from_ann(i,:) = M_forward_model;
k=1;
g=1;
g2=1;
else
% Derivatives

```

```

dM_dx1 = 0;
dM_dx2 = 0;
dM_dx3 = 0;
for l = 1:size(u,2)
dM_dx1 = dM_dx1 + v(l,1)*z(l,1)*(1-z(l,1))*u(1,1);
dM_dx2 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(2,1);
dM_dx3 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(3,1);
end
dM_dX(:,1) = dM_dx1;
dM_dX(:,2) = dM_dx2;
dM_dX(:,3) = dM_dx3;
X_assume = X_assume - 0.1*dM_dX;
X_assume(:,3) = round(X_assume(:,3));
itr(i,1) = itr(i,1) +1;
if X_assume(1)<0.05 || X_assume(1)>2.00||X_assume(2)<0.25
    || X_assume(2)>1.3 || X_assume(3)<2 || X_assume(3)>15 || alph > 0.05
k=1;
itr(i,1)=0;
end end end end
g1 = 0;
while(g1 == 0)
k=0;
while(k == 0)
M_forward_model = (rho.*pi.*(X_assume(:,1).^2)/4).*(sqrt(1
    + (X_assume(:,3).*pi.*X_assume(:,2)).^2) + 2);
z = z_value(X_assume);
alph = abs(M_given(i,1) - M_forward_model);

```

```

if alph < 0.0001    % accepting close amplitudes
X_accept(i,:) = X_assume;
M_from_ann(i,:) = M_forward_model;

k=1;
g1=1;
else
% Derivatives
dM_dx1 = 0;
dM_dx2 = 0;
dM_dx3 = 0;
for l = 1:size(u,2)
dM_dx1 = dM_dx1 + v(l,1)*z(l,1)*(1-z(l,1))*u(1,l);
dM_dx2 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(2,l);
dM_dx3 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(3,l);
end
dM_dX(:,1) = dM_dx1;
dM_dX(:,2) = dM_dx2;
dM_dX(:,3) = dM_dx3;
if s==0
g(r,:) = dA_dX;
h(r,:) = -g(r,:) ;
X_assume = X_assume + 0.1*h(r,:);
s=1;
r=r+1;
else
g(r,:) = dA_dX;

```

```

gamma(r-1,:) = (g(r,:)*g(r,:))'./(g(r-1,:)*g(r-1,:))';
h(r,:) = - g(r,:) + gamma(r-1,:)*h(r-1,:);
X_assume = X_assume + 0.1*h(r,:);
r = r+1;
end
if X_assume(1)<0.05 || X_assume(1)>2.00||X_assume(2)<0.25
    || X_assume(2)>1.3 || X_assume(3)<2 || X_assume(3)>15 || alph > 2
k=1;
r=2;
end end end end end end
for i=1:ns
M(i,1) = (rho.*pi.*(X_accept(i,1).^2)/4).*(sqrt(1
    + (X_accept(i,3).*pi.*X_accept(i,2)).^2) + 2);
end
E=0;
for i=1:ns
E = E + abs(M(i,1)-M_given(i,1));
end
Error = E
E1=0;
for i=1:ns
E1 = E1 + abs((M(i,1)-M_given(i,1))/M(i,1));
end
Error_normalised = E1/ns

```

D.4 Ten Bar Truss Problem

```
minc=[0.01 0.01 1]; %min beta update value
maxc=[1.9 1 12]; % max beta update value
ratiomin=minc./maxc; %lower limit for ratio
ratiomax=maxc./minc; %upper limit for ratio
ns = 200;

ulim_x1=2.00; %max value
llim_x1=0.05; %min value
ulim_x2=1.30; %max value
llim_x2=0.25; %min value
ulim_x3=15; %max value
llim_x3=2.00; %min value
rho = 7845/(1e6); % kg/m.^3
iterations=load('iterations.txt');
M_given=load('Mass.txt');
u = transpose(load('weights_u.dat'));
v = load('weights_v.dat');
p = load('weights_p.dat');
q = load('weights_q.dat');
for i = 1:ns
    r=1;
    s=0;
    itr(i,1)=0;
    g2 = 0;
    if itr(i,1) <= (iterations(i,1)/2)
        while(g2 == 0)
```



```

x1 = round(100*(llim_x1+(ulim_x1-llim_x1)*rand))/100;
x2 = round(100*(llim_x2+(ulim_x2-llim_x2)*rand))/100;
x3 = randi([llim_x3 ulim_x3],1,1);
X_assume(:,1) = x1;
X_assume(:,2) = x2;
X_assume(:,3) = x3;
k=0;
while(k == 0)
M_forward_model = (rho.*pi.*(X_assume(:,1).^2)/4).*(sqrt(1
    + (X_assume(:,3).*pi.*X_assume(:,2)).^2) + 2);
z = z_value(X_assume);
alph = abs(M_given(i,1) - M_forward_model);
if alph < 0.0001    % accepting close amplitudes
X_accept(i,:) = X_assume;
M_from_ann(i,:) = M_forward_model;
k=1;
g=1;
g2=1;
else
% Derivatives
dM_dx1 = 0;
dM_dx2 = 0;
dM_dx3 = 0;
for l = 1:size(u,2)
dM_dx1 = dM_dx1 + v(l,1)*z(l,1)*(1-z(l,1))*u(1,l);
dM_dx2 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(2,l);
dM_dx3 = dM_dx3 + v(l,1)*z(l,1)*(1-z(l,1))*u(3,l);

```

```

end

dM_dX(:,1) = dM_dx1;
dM_dX(:,2) = dM_dx2;
dM_dX(:,3) = dM_dx3;

X_assume = X_assume - 0.1*dM_dX;

X_assume(:,3) = round(X_assume(:,3));

itr(i,1) = itr(i,1) +1;

if X_assume(1)<0.05 || X_assume(1)>2.00||X_assume(2)<0.25
    || X_assume(2)>1.3 || X_assume(3)<2 || X_assume(3)>15 || alph > 0.05
k=1;

itr(i,1)=0;

end end end end

g1 = 0;

while(g1 == 0)

k=0;

while(k == 0)

M_forward_model = (rho.*pi.*(X_assume(:,1).^2)/4).*(sqrt(1
    + (X_assume(:,3).*pi.*X_assume(:,2)).^2) + 2);

z = z_value(X_assume);

alph = abs(M_given(i,1) - M_forward_model);

if alph < 0.0001    % accepting close amplitudes
X_accept(i,:) = X_assume;

M_from_ann(i,:) = M_forward_model;

k=1;

g1=1;

else

% Derivatives

```

```

dM_dx1 = 0;
dM_dx2 = 0;
dM_dx3 = 0;
for l = 1:size(u,2)
dM_dx1 = dM_dx1 + v(l,1)*z(l,1)*(1-z(l,1))*u(1,1);
dM_dx2 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(2,1);
dM_dx3 = dM_dx2 + v(l,1)*z(l,1)*(1-z(l,1))*u(3,1);
end
dM_dX(:,1) = dM_dx1;
dM_dX(:,2) = dM_dx2;
dM_dX(:,3) = dM_dx3;
if s==0
g(r,:) = dA_dX;
h(r,:) = -g(r,:) ;
X_assume = X_assume + 0.1*h(r,:);
s=1;
r=r+1;
else
g(r,:) = dA_dX;
gamma(r-1,:) = (g(r,:)*g(r,:)')./(g(r-1,:)*g(r-1,:)');
h(r,:) = - g(r,:) + gamma(r-1,:)*h(r-1,:);
X_assume = X_assume + 0.1*h(r,:);
r = r+1;
end
if X_assume(1)<0.05 || X_assume(1)>2.00||X_assume(2)<0.25
|| X_assume(2)>1.3 || X_assume(3)<2 || X_assume(3)>15 || alph > 2
k=1;

```

```

r=2;

end end end end end end

for i=1:ns
M(i,1) = (rho.*pi.*(X_accept(i,1).^2)/4).*(sqrt(1 + (X_accept(i,3)
        .*pi.*X_accept(i,2)).^2) + 2);

end

E=0;

for i=1:ns
E = E + abs(M(i,1)-M_given(i,1));

end

Error = E

E1=0;

for i=1:ns
E1 = E1 + abs((M(i,1)-M_given(i,1))/M(i,1));

end

Error_normalised = E1/ns

```